UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

ANALYSE ET AMÉLIORATION DE LA QUALITÉ DE SERVICES WEB MULTIMÉDIA ET LEURS MISES EN ŒUVRE SUR ORDINATEUR ET SUR FPGA

Thèse de doctorat
Spécialité : génie électrique

Amer AL-CANAAN

Jury : Ahmed KHOUMSI (Directeur)
Soumaya CHERKAOUI (Rapporteur)
Lucien OUEDRAOGO (Examinateur externe)
Abdelhamid MAMMERI (Examinateur externe)

Sherbrooke (Québec) Canada Décembre 2014

Ce travail est dédié à ma très chère mère décédée ainsi qu'à mon père et à ma famille

RÉSUMÉ

Les services Web, issus de l'avancée technologique dans le domaine des réseaux informatiques et des dispositifs de télécommunications portables et fixes, occupent une place primordiale dans la vie quotidienne des gens. La demande croissante sur des services Web multimédia (SWM), en particulier, augmente la charge sur les réseaux d'Internet, les fournisseurs de services et les serveurs Web. Cette charge est essentiellement due au fait que les SWM de haute qualité nécessitent des débits de transfert et des tailles de paquets importants. La qualité de service (par définition, telle que vue par l'utilisateur) est influencée par plusieurs facteurs de performance, comme le temps de traitement, le délai de propagation, le temps de réponse, la résolution d'images et l'efficacité de compression.

Le travail décrit dans cette thèse est motivé par la demande continuellement croissante de nouveaux SWM et le besoin de maintenir et d'améliorer la qualité de ces services. Nous nous intéressons tout d'abord à la qualité de services (QdS) des SWM lorsqu'ils sont mis en œuvre sur des ordinateurs, tels que les ordinateurs de bureau ou les portables. Nous commençons par étudier les aspects de compatibilité afin d'obtenir des SWM fonctionnant de manière satisfaisante sur différentes plate-formes. Nous étudions ensuite la QdS des SWM lorsqu'ils sont mis en œuvre selon deux approches différentes, soit le protocole SOAP et le style RESTful. Nous étudions plus particulièrement le taux de compression qui est un des facteurs influençant la QdS.

Après avoir considéré sous différents angles les SWM avec mise en œuvre sur des ordinateurs, nous nous intéressons à la QdS des SWM lorsqu'ils sont mis en œuvre sur FPGA. Nous effectuons alors une étude et une mise en œuvre qui permet d'identifier les avantages à mettre en œuvre des SWM sur FPGA.

Les contributions se définissent en cinq volets comme suit :

1. Nous introduisons des méthodes de création, c'est-à-dire conception et mise en œuvre, de SWM sur des plate-formes logicielles hétérogènes dans différents environnements tels que Windows, OS X et Solaris. Un objectif que nous visons est de proposer une approche permettant d'ajouter de nouveaux SWM tout en garantissant la compatibilité entre les plate-formes, dans le sens où nous identifions les options nous permettant d'offrir un ensemble riche et varié de SWM pouvant fonctionner sur les différentes plate-formes.

2. Nous identifions une liste de paramètres pertinents influençant la QdS des SWM mis en œuvre selon le protocole SOAP et selon le style REST.

3. Nous développons un environnement d'analyse pour quantifier les impacts de chaque paramètre identifié sur la QdS de SWM. Pour cela, nous considérons les SWM mis en œuvre selon le protocole SOAP et aussi selon style REST. Les QdS obtenues avec SOAP et REST sont comparées objectivement. Pour faciliter la

comparaison, la même gamme d'images (dans l'analyse de SWM SOAP) a été réutilisée et les mêmes plate-formes logicielles.

4. Nous développons une procédure d'analyse qui permet de déterminer une corrélation entre la dimension d'une image et le taux de compression adéquat. Les résultats obtenus confirment cette contribution propre à cette thèse qui confirme que le taux de compression peut être optimisé lorsque les dimensions de l'image ont la propriété suivante : le rapport entre la longueur et la largeur est égal au nombre d'or connu dans la nature. Trois libraires ont été utilisées à savoir JPEG, JPEG2000 et DjVu.
5. Dans un volet complémentaire aux quatre volets précédents, qui concernent les SWM sur ordinateurs, nous étudions ainsi la conception et la mise en œuvre de SWM sur FPGA. Nous justifions l'option de FPGA en identifiant ses avantages par rapport à deux autres options : ordinateurs et ASICs. Afin de confirmer plusieurs avantages identifiés, un SWM de QdS élevée et de haute performance est créé sur FPGA, en utilisant des outils de conception gratuits, du code ouvert (*open-source*) et une méthode fondée uniquement sur HDL. Notre approche facilitera l'ajout d'autres modules de gestions et d'orchestration de SWM.
6. La mise à jour et l'adaptation du code open-source et de la documentation du module *Ethernet IP Core* pour la communication entre le FPGA et le port Ethernet sur la carte Nexys3. Ceci a pour effet de faciliter la mise en œuvre de SWM sur la carte Nexys3.

Mots-clés : services Web multimédia, paramètres de qualité de service, service Web RESTful, service Web SOAP, paramètres de performance, compression binaire, FPGA.

ABSTRACT

Web services, which are the outcome of the technological advancements in IT networks and hand-held mobile devices for telecommunications, occupy an important role in our daily life. The increasing demand on multimedia Web services (MWS), in particular, augments the load on the Internet, on service providers and Web servers. This load is mainly due to the fact that the high-quality multimedia Web services necessitate high data transfer rates and considerable payload sizes. The quality of service (QoS, by definition as it is perceived by the user) is influenced by several factors, such as processing time, propagation delay, response time, image resolution and compression efficacy.

The research work in this thesis is motivated by the persistent demand on new MWS, and the need to maintain and improve the QoS. Firstly, we focus on the QoS of MWS when they are implemented on desktop and laptop computers. We start with studying the compatibility aspects in order to obtain MWS functioning satisfactorily on different platforms. Secondly, we study the QoS for MWS implemented according to the SOAP protocol and the RESTful style. In particular, we study the compression rate, which is one of the pertinent factors influencing the QoS.

Thirdly, after the study of MWS when implemented on computers, we proceed with the study of QoS of MWS when implemented on hardware, in particular on FPGAs. We achieved thus comprehensive study and implementations that show and compare the advantages of MWS on FPGAs.

The contributions of this thesis can be resumed as follows:

1. We introduce methods of design and implementation of MWS on heterogeneous platforms, such as Windows, OS X and Solaris. One of our objectives is to propose an approach that facilitates the integration of new MWS while assuring the compatibility amongst involved platforms. This means that we identify the options that enable offering a set of rich and various MWS that can run on different platforms.

2. We determine a list of relevant parameters that influence the QoS of MWS.

3. We build an analysis environment that quantifies the impact of each parameter on the QoS of MWS implemented on both SOAP protocol and RESTful style. Both QoS for SOAP and REST are objectively compared. The analysis has been held on a large scale of different images, which produces a realistic point of view describing the behaviour of real MWS.

4. We develop an analysis procedure to determine the correlation between the aspect ratio of an image and its compression ratio. Our results confirm that the compression ratio can be improved and optimised when the aspect ratio of

an image is close to the golden ratio, which exists in nature. Three libraries of compression schemes have been used, namely: JPEG, JPEG2000 and DjVu.

5. Complementary to the four contributions mentioned above, which concern the MWS on computers, we study also the design and implementation of MWS on FPGA. This is justified by the numerous advantages that are offered by FPGAs, compared to the other technologies such as computers and ASICs. In order to highlight the advantages of implementing MWS on FPGA, we developed on FPGA a MWS of high performance and high level of QoS. To achieve our goal, we utilised freely available design utilities, open-source code and a method based only on HDL. This approach is adequate for future extensions and add-on modules for MWS orchestration.

Keywords: multimedia Web services, Quality of service parameters, performance parameters, RESTful Web service, SOAP Web service, binary compression, FPGA.

REMERCIEMENTS

Remerciements

Je tiens à remercier chaleureusement mon directeur de thèse, Monsieur Ahmed KHOUMSI, Professeur à l'Université de Sherbrooke (Canada), pour son appui, son support et ses conseils bien appréciés. Je tiens à remercier également Madame Soumaya CHERKAOUI, Professeure à l'Université de Sherbrooke (Canada) pour avoir accepté d'être rapporteur de cette thèse, Monsieur Lucien OUEDRAOGO, Concepteur à « *The Mathworks, Natick, Massachussetts, USA* »(aux États Unis), qui a accepté d'examiner ce travail, ainsi que Monsieur Abdelhamid MAMMERI, Chercheur à l'Université d'Ottawa, d'avoir accepté d'être correcteur de cette thèse.

Je voudrais également remercier mon collègue Hicham CHAKIB, qui occupait une position de professeur en génie électrique à l'université King Saud à Riyadh (Arabie Saoudite) pour son amitié et son aide. Je remercie tous les collègues et les membres du département de génie électrique et de génie informatique. Plus particulièrement, je remercie également Madame Danielle GAGNÉ secrétaire du département de génie électrique et de génie informatique pour son aide, sa sympathie et sa disponibilité. Je tiens également à remercier Monsieur Sylvain LAUZIER du groupe de support en génie électrique pour sa disponibilité et son aide précieuse. Mes remerciements vont également à Madame Nathalie MÉNARD au poste de commis aux affaires académiques, 2^e et 3^e cycles, à Madame Geneviève SIMARD au poste de soutien de la direction de la faculté de Génie et Madame Linda SIMONCELLI au poste de commis aux affaires académiques pour leurs excellent accueil, communication et suivi de mon dossier étudiant.

Je tiens aussi à remercier toute ma famille, en particulier, mes parents, mon épouse et mes enfants, pour leur patience, leur encouragement et leur soutien.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LEXIQUE

Terme technique	Définition
API	Application Programming Interface
BPEL4WS	Business Process Execution Language for Web Services
CPL	Call Processing Language
CPU	Central Processing Unit
DAML-S	DARPA Agent Markup Language-Service
DAML-S	DARPA Agent Markup Language for Services
FPGA	Fiels Programmable Array
FSM	Finite State Machine
GIF	Graphics Interchange Format
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IS	Interaction(s) de Services
Java-RMI	Java-(Remote Method Invocation)
JWSDP	Java Web Service Development Package
MEF	Machine à État Fini
MIME	Multipurpose Internet Mail Extensions
ORP	Object Request broker Protocol
QdS	Qualité de Service
REST	REpresentational State Transfer
RMI/IIOP	Remote Methode Invocation/Internet Inter-ORP
RPC	Remote Procedure Call
ROA	Resource Oriented Architecture
RTP	Real Time Protocol
RTSP	Real Time Streaming Protocol
RTT	Round-Trip Time
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SMS	Short Messaging Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol

Terme technique	Définition
SW	Services Web
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WSLA	Web Service Level Agreement
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation

LISTE DES SYMBOLES

Symbole	Définition
ϕ	Golden ratio (1.61803399)
w	largeur d'une image
h	longueur d'une image

LISTE DES ACRONYMES

Acronyme	Définition
JAIN-SIP	Java Application Interface for Integrated Networks using SIP
SIP	Session Initiation Protocol
Phy	Physical interface transceiver

CHAPITRE 1

INTRODUCTION

Avant de préciser les motivations, les objectifs et les contributions de notre thèse, nous commencerons par présenter les sujets relatifs à notre travail, à savoir les services Web multimédia (SWM) et la qualité de service (QdS) dans de tels services.

1.1 Services Web multimédia

Les services Web [29, 54, 75, 99] (SW) sont des services logiciels accessibles à travers Internet en utilisant des protocoles communs sur Internet, comme HTTP[1], XML et HTML. Les SW offrent une portée et une accessibilité plus importante comparativement aux solutions adaptées pour les systèmes répartis [41], comme Java-RMI et CORBA, puisque les SW sont fondés sur HTTP et XML.

La création de SWM suit l'une des deux architectures suivantes : SOA (*Service Oriented Architecture*) [35] et ROA (*Resources Oriented Architecture*). L'architecture SOA est représentée par les protocoles SOAP (*Simple Object Access Protocol*) et WSDL (en anglais, *Web Services Description Language*) dont discute la section 1.1.2. L'architecture ROA est caractérisée par le style dit REST (*REpresentational State Transfer*) qui est présenté dans la section 1.1.1.

1.1.1 Services Web mis en œuvre selon le style REST

Le terme REST [27, 120] décrit un style d'architecture logicielle pour des systèmes répartis, en particulier pour le Web. Ce terme fut introduit par Roy Fielding, l'un des auteurs principaux de la spécification de HTTP. Dans certains textes, le terme *RESTful* [95] désigne les systèmes qui suivent les principes de REST. Le Web est un bon exemple d'un tel système où HTTP constitue une interface standard permettant aux clients et aux

[1] HTTP [78] est un protocole qui permet de transférer des informations, comme les fichiers et les pages Web, offerts à travers le Web. Le Web est l'une des sources d'informations (comme WAIS, Archie, Gopher, INDIE, Prospro et X.500 [103]) disponibles sur Internet.

serveurs de communiquer et d'échanger des représentations des ressources. Les identificateurs uniformes d'informations ou URIs (*Uniform Resource Identifiers*) constituent un concept principal pour le Web et pour le style REST.

Les applications sur le Web utilisent principalement le protocole HTTP pour transférer des informations de types multiples (en anglais, *Content Types*), comme HTML, ainsi que d'autres services supplémentaires comme DNS (en anglais, *Domain Name Service*) pour diriger les paquets de données. Le protocole HTML permet d'engendrer des images, des vidéos, des codes Javascript et des applets Java. Le protocole HTTP impose une interface uniforme pour accéder aux URIs des ressources offerts sur Internet. Les différentes techniques de *caches* [100] contribuent à l'amélioration de la performance de HTTP pour certains cas. Le protocole HTTP expose ses commandes (ou méthodes) selon des en-têtes et des contenus identifiés par un type de contenu spécifique MIME (en anglais, *Multipurpose Internet Mail Extensions*). Les méthodes HTTP les plus importantes incluent : POST (créer et rendre publique), GET (réclamer), PUT (mettre à jour) et DELETE (supprimer). Ces méthodes sont des synonymes aux commandes CREATE, READ, UPDATE et DELETE (CRUD), utilisées dans le domaine de bases de données.

Une application exploitant le protocole HTTP est dit RESTful quand elle fonctionne en mode sans-état (en anglais, *stateless*), où chaque message HTTP contient des informations complètes pour son traitement. Dans ce cas là, les clients et les serveurs ne maintiennent aucun état particulier. Si l'application ou le service exige un état lors de son fonctionnement, l'état peut être exposé selon un URI et traité comme une ressource. Par ailleurs, les services non RESTful favorisent l'utilisation des témoins (en anglais, *cookies*) permettant à HTTP de maintenir l'état de messages échangés (en anglais, *stateful*).

Les SW construits selon le style REST disposent des caractéristiques spécifiques comme suit :

- Chaque fonction et son état sont distincts dans l'application, comme deux ressources distinctes.
- Chaque ressource possède un identificateur uniforme ou URI spécifique.
- Toutes les ressources partagent une interface commune pour exposer les états d'une ressource à un client selon des opérations bien définies et un ensemble de types de contenus.
- Le protocole HTTP effectue les transferts de données en offrant des propriétés particulières comme suit :

- ☐ Offrir un modèle d'interaction de type client-serveur.
- ☐ Effectuer un mode de communication sans-état. Les messages envoyés entre le client et le serveur contiennent toutes les informations nécessaires pour les traiter. C'est-à-dire que chaque requête est autonome.
- ☐ Profiter du support des mémoires *caches* pour augmenter l'efficacité.
- ☐ Profiter de plusieurs entités intermédiaires dans le réseau. Par exemple, des serveurs *proxies*, des passerelles et des pare-feux.

Une application accède à une ressource sachant son emplacement ou URI, l'action à effectuer et le format de l'information, c'est-à-dire sa représentation. Parmi les applications dites *RESTful* [120], nous citons *WebDAV* (en anglais, *Web-based Distributed Authoring and Versioning*) qui est fondée sur HTTP. D'autres exemples existent comme *blogsphere* qui est l'univers de *weblogs*, *Ruby on Rails 1.2* [95] et *Atom Publishing Protocol* pour publier les *blogs*. Par exemple, *WebDAV* permet l'accès à des informations d'un répertoire électronique selon une interface Web comme suit :

- vérifier si l'interface Web est valide et fonctionnelle ;
- vérifier et gérer les mises-à-jour effectuées.

WebDAV exploite la requête *HEAD* de HTTP et définit d'autres méthodes HTTP spécifiques à lui comme PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK et UNLOCK.

Certaines entités et protocoles informatiques ont des caractéristiques dites *RESTful* [95], comme :

- Le protocole *Modbus* de communications en série pour les systèmes embarqués qui sert à manipuler des blocs de mémoires dans les PLCs (*Programmable Logic Controllers*). L'écriture et la lecture des blocs de mémoire s'effectuent de façon similaire aux méthodes GET et PUT.
- *Java Beans* et d'autres systèmes qui effectuent principalement des modifications de propriétés suivant les commandes GET et PUT de REST au lieu des codes d'écriture d'objets.
- Les deux utilitaires *couper* et *coller* du bureau du travail (en anglais, *desktop*) qui ont une approche similaire à celle de REST. Ces utilitaires sont par exemple utilisés pour la localisation de la souris sur l'écran et les sélections sur-lignées qui utilisent peu de verbes comme : CUT, COPY et PASTE.

Les avantages du style REST [58, 108, 120] se résument comme suit :

- Le temps de réponse et le débit de traitement profitent de la disposition de mémoires caches (en anglais, *caching*) pour HTTP.

- L'aspect sans-état ne requiert pas de maintenance de l'état lors d'une transaction donnée, ce qui améliore la possibilité d'étendre la structure de l'application (en anglais, *scalability*) permettant à plusieurs serveurs d'effectuer du traitement aux requêtes, par exemple.
- Une approche directe c'est-à-dire que le code à élaborer est minime comparé à ceux d'autres approches. Par conséquent, REST n'est pas dépendant d'APIs ni de logiciels spécifiques.
- Il donne des résultats équivalents à ceux des autres approches.
- Il offre une compatibilité à long terme comparativement à RPC du fait que :
  - ☐ les types de documents comme HTML évoluent sans affecter les anciennes versions et
  - ☐ les ressources peuvent ajouter de nouveaux types de contenus sans influencer les anciens types.

1.1.2 Services Web mis en œuvre selon le protocole SOAP

L'architecture des SW selon le protocole SOAP est typiquement constituée de trois entités indépendantes et réparties : Le fournisseur, le registre et le client, tels que montrés sur la figure 1.1.

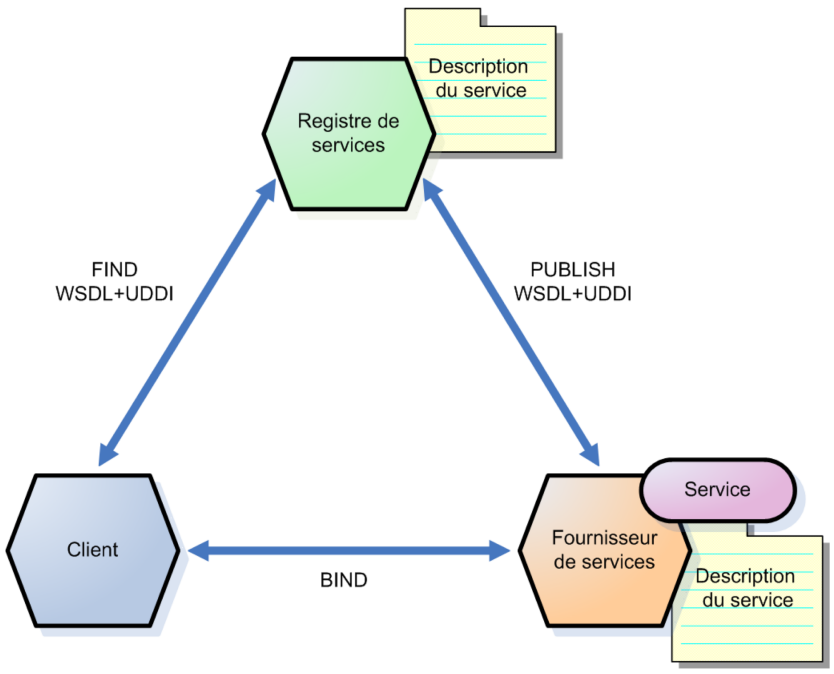


Figure 1.1 L'architecture SOA pour les services Web

L'intégration des trois entités est assurée par les langages et les protocoles suivants :

- XML est un langage populaire facile à interpréter par l'homme et par la machine. XML permet de représenter les données d'une manière standard et indépendante de plate-formes.
- SOAP est un protocole de communication entre les applications sur le Web. SOAP est fondé sur XML et HTTP,
- WSDL est un langage fondé sur XML pour décrire l'interface d'un service Web de façon abstraite [114]. WSDL est nécessaire pour l'identification, la description, la découverte et l'exploitation d'un service Web.
- UDDI (en anglais, *Universal Description, Discovery and Integration*) [112] est un registre de SW qui sert à promouvoir les descriptions en WSDL des services. L'UDDI offre un API permettant aux clients et aux fournisseurs de manipuler les descriptions de services.

Une fois un service implémenté, le fournisseur procède à le promouvoir pour des clients potentiels. Un scénario de création et d'exploitation d'un service Web se présente comme suit :

1. Le fournisseur implémente son service Web selon les interfaces standardisées pour le Web. Il publie une description de son service en WSDL qui est déposée dans le registre UDDI. Une fois cette phase terminée, le service est disponible et prêt pour des clients potentiels.

2. Le registre est une entité intermédiaire entre le fournisseur de services et le client. Il fait la promotion des descriptions WSDL des SW. Le registre classifie le nouveau service dans ses répertoires (bases de données) afin de faciliter les recherches effectuées par des clients, à savoir que l'UDDI est publique et accessible à travers le Web.

3. Le client désirant utiliser un service Web, effectue une recherche selon des mots-clés pertinents dans un registre UDDI, à l'aide d'un API spécifique de ce dernier. Le registre lui envoie une liste des descriptions de tous les SW correspondants.

4. Le client choisit l'un des services retrouvés dans l'UDDI et commence à exploiter le service en envoyant directement des messages (requêtes) SOAP à l'aide des informations fournies par le document WSDL. À noter que toutes les communications entre les 3 joueurs : fournisseur, client et UDDI, se font en SOAP qui est transporté sur HTTP.

Le protocole SOAP permet la communication entre des applications sur le Web. Étant basé sur XML et le protocole de transport HTTP, SOAP [42] offre des caractéristiques intéressantes qui se résument comme suit :

- Il facilite l'échange de messages entre les applications sur des plates-formes multiples, vu qu'il est transporté sur HTTP, déjà supporté à travers le Web. SOAP sert à encoder des messages et à définir des messages RPC en XML. Un message SOAP est composés de deux parties : le contenu et l'en-tête sous forme XML. Les applications communiquent généralement entre elles à l'aide du RPC (*Remote Procedure Call*), ce qui impose une barrière de compatibilité et de sécurité puisque RPC pourrait être bloqué par plusieurs serveurs et *proxies*. Par contre, HTTP est supporté par tous les serveurs et les applications Web.
- SOAP ne dépend ni des systèmes d'exploitation ni des technologies particulières des langages de programmation.

WSDL [115] est un langage fondé sur XML servant à décrire les interfaces de services offerts par des fournisseurs de SW, ce qui permet aux clients de découvrir ces services en effectuant des recherches selon des critères diversifiés. WSDL est inspiré de SOAP et de NASSL (en anglais, *Network Accessible Service Specification Language*) d'IBM [114]. WSDL et SOAP étant fondés sur XML, ils utilisent des *namespaces* afin d'ajouter des informations supplémentaires dans le but de définir leurs éléments standards. Ces éléments standards sont ainsi interprétés d'une manière spécifique. L'interaction entre SOAP et WSDL peut être résumée comme suit : WSDL offre un mécanisme décrivant un ensemble d'appels de fonctions (opérations) à distance sous forme d'un port accessible par un échange de messages SOAP [130].

WSDL décrit les SW de deux manières : abstraite et concrète. WSDL 2.0 décrit les SW à l'aide de termes et d'éléments spécifiques, en particulier nous avons les éléments suivants :

Operation est une association de messages avec une certaine séquence (en anglais, *pattern*) d'échange de messages.

Interface sert à regrouper des *opérations* sans être reliée à un moyen de transport.

Binding spécifie les détails du transport pour les *interfaces*.

Service regroupe des *points terminaux* qui implémentent une *interface* commune.

Point Terminal (en anglais, *end-point*) associe une adresse réseau à un *binding*.

La notion de *namespaces* est souvent utilisée dans le contexte de SW pour décrire l'emplacement des documents standardisés qui réglementent le format d'un document WSDL (comme un schéma XML).

La balise principale dans un document WSDL 2.0 est appelée *Definitions* dans laquelle se trouvent d'autres composants catégorisés comme suit :

- Des composants WSDL représentés par les éléments : *interfaces*, *bindings* (relieurs) et *services*.
- Des composants spécifiques appelés composants d'un système-type (en anglais, *type system components*) représentant les déclarations reliées à un système-type. C'est-à-dire que ce sont des déclarations d'éléments reliés au service et à sa mise en œuvre. Ces composants sont représentés par l'élément *types*.

Les propriétés du composant principal *Definitions* d'un document WSDL se résument comme suit :

- {*interfaces*} représentent un ensemble d'éléments de type `interface`[2]. Elles correspondent à tous les éléments ayant le nom `<interface>` (sous-éléments de `definitions`) et leurs définitions qui sont incluses ou importées.
- {*bindings*} représentent un ensemble d'éléments de type `binding`. Elles correspondent à tous les éléments ayant le nom `<binding>` (sous-éléments de `definitions`) et leurs définitions sont incluses ou importées.
- {*services*} représentent un ensemble d'éléments de type `service`. Elles correspondent à tous les éléments ayant le nom `<service>` (sous-éléments de `definitions`) et leurs définitions soient incluses ou importées.
- {*element declarations*} représentent un ensemble de définitions d'éléments nommés dans un document WSDL. Elles correspondent à tous les éléments ayant le nom `<types>` (sous-éléments de `definitions`) et leurs définitions importées.

Par exemple, les lignes suivantes représentent un squelette d'un document WSDL :

```
<definitions>
  <binding
       name="xs\ ! :NCName"
       interface="xs\ ! :QName"' >
    <documentation />'
    [ <fault /> | <operation /> | <feature /> | <property /> ]*
  </binding>
</definitions>
```

[2]Une *interface* regroupe des éléments de type *operation* qui décrivent l'échange de messages d'un service Web.

Dans un document WSDL, les éléments sont, soit inclus explicitement, soit importés depuis un URI de façon implicite en utilisant une référence. Les sous-éléments qui ont un `targetNamespace` sont inclus sous un même composant (élément parent comme `définitions`). C'est-à-dire que l'attribut `targetNamespace` regroupe un ensemble de définitions de composants, d'éléments et de sous-éléments reliés.

Les lignes suivantes montrent la description en WSDL d'un service Web (agence de voyage *TicketAgent*) [115]. L'attribut *targetNamespace* décrit l'URI qui définit les sous-éléments dans le document WSDL. L'élément `operation` décrit deux opérations, réclamer une liste de vols et réserver un billet, qui sont reliées avec une seule interface. Les déclarations du service se trouvent sur un URI dans l'élément `types` :

```
<'xml version="1.0" encoding="UTF-8"'>
<wsdl :definitions
 targetNamespace="http ://example.org/TicketAgent.wsdl20"
 xmlns :xsTicketAgent="http ://example.org/TicketAgent.xsd"
 xmlns :wsdl="http ://www.w3.org/2004/03/wsdl"
 xmlns :xs="http ://www.w3.org/2001/XMLSchema"
 xmlns :xsi="http ://www.w3.org/2001/XMLSchema-instance"
 xsi :schemaLocation="http ://www.w3.org/2004/03/wsdl wsdl20.xsd">

  <wsdl :types>
     <xs :import schemaLocation="TicketAgent.xsd"
                 namespace="http ://example.org/TicketAgent.xsd"/>
  </wsdl :types>

  <wsdl :interface name="TicketAgent">
    <wsdl :operation name="listFlights"
                  pattern="http ://www.a.org/wsdl/in-out">
       <wsdl :input element="xsTicketAgent :listFlightsRequest"/>
       <wsdl :output element="xsTicketAgent :listFlightsResponse"/>
    </wsdl :operation>

    <wsdl :operation name="reserveFlight"
                  pattern="http ://www.a.org/wsdl/in-out">
      <wsdl :input element="xsTicketAgent :reserveFlightRequest"/>
      <wsdl :output element="xsTicketAgent :reserveFlightResponse"/>
    </wsdl :operation>
  </wsdl :interface>
</wsdl :definitions>
```

1.2 Qualité de services Web multimédia

La qualité des services (QdS) décrit le niveau de satisfaction du client [94] lors de l'utilisation d'un SW. La nuance entre la performance et la QdS se présente en soulignant que la performance correspond au point de vue technique, qui n'est pas nécessairement ressenti par l'utilisateur ou le client. Par exemple, une mise à jour de serveurs plus rapides et l'ajout de mémoire plus grande pour servir un réseau de télécommunication IP n'a pas toujours un effet positif sur un client qui utilise son téléphone à l'occasion. Dans ce cas-ci, la performance augmente l'appréciation de l'ensemble de clients en produisant des services plus attirants et plus intéressants pour des clients potentiels. Par ailleurs, un client peut être impressionné par la couleur de son appareil et le côté esthétique plus que la qualité du son lors des appels, par exemple.

La QdS d'un service ou d'un produit, en particulier un service Web, est définie par un ensemble de paramètres techniques influençant le taux de satisfaction du client lorsqu'il utilise le produit ou le service. De tels paramètres seront appelés : paramètres de QdS. Les exigences en QdS se traduisent donc en des contraintes sur les valeurs pouvant être prises par ces paramètres de QdS.

1.3 Problématique, Motivation et Objectifs

1.3.1 Problématique et motivation

La demande croissante sur des services Web multimédia (SWM) augmente la pression sur les fournisseurs de services Web (SW) pour maintenir un niveau acceptable de qualité de service (QdS). Les SWM se distinguent par le contenu multimédia et par des exigences particulières. Par exemple, l'entête des paquets d'un SWM constitue une fraction minimale comparativement au contenu multimédia incluant la voix, les images et la vidéo. Les exigences particulières du contenu multimédia incluent, entre autres, des contraintes sur la compression, le débit adéquat pour le transfert, les protocoles de transfert favorisant la diffusion de messages multimédia et un niveau acceptable de QdS.

Les domaines de recherches de SWM peuvent se présenter en deux catégories :

1. Conception de SWM sur des systèmes logiciels : Des solutions sont proposées à plusieurs niveaux, tels que le niveau des couches réseau, transport et application [32, 62, 99, 100, 108] du modèle OSI.

2. Conception de SWM sur des systèmes matériels : Les ASICs figurent dans cette catégorie, mais nous avons opté de les écarter de notre étude en raison de leur coût élevé et du temps considérable de leur mise en œuvre. Nous avons plutôt opté d'étudier la mise en œuvre sur FPGA [36, 97], ce qui exige une bonne connaissance des portes logiques et des langages Verilog et VHDL.

Malgré les travaux de recherche effectués pour la conception de SWM sur ordinateurs et sur FPGA, les solutions apportées pour l'amélioration de la QdS de SWM ne sont généralement pas satisfaisantes pour les SWM. Ceci est dû à la nature du Web qui n'est pas profondément conçu pour offrir une QdS soutenue à des SWM nécessitant des contraintes particulières telles que les suivantes :

- Un débit de transfert adéquat pour des données multimédia. Selon [50, 55], le débit standard pour des séquences vidéos de qualité varie entre $50\ K$bps[3] et $8\ M$bps pour des dimensions autour de 176×144 pixels et de 1920×1080 pixels, respectivement.
- Une taille importante d'information. La taille d'une trame vidéo non comprimée s'élève à $460.8\ K$bits (pour une séquence vidéo de 160×120 pixels). Les images numériques prises depuis des téléphones cellulaires ou des appareils photos numériques dépassent facilement $2\ M$ octets.

Les faiblesses des travaux de recherches existants sur les SWM se résument comme suit :

1. L'utilisation de tailles non représentatives de données multimédia. La majorité des travaux de recherches n'utilisent que des modèles simplifiés de SW qui ne décrivent pas adéquatement les SWM. Ces derniers se distinguent des SW de base sur deux aspects, soit le débit et la taille importants du contenu riche en son, en images et en vidéo. Par exemple, la taille de données utilisée dans [126] est fixée à $980\ K$ octets comparativement à notre approche qui inclut des tailles variées jusqu'à $4.617\ M$ octets. Dans [99], les messages échangés étaient de type *HelloWorld* (une ligne de texte qui ne dépasse pas 200 octets, y compris l'entête IP) pour mesurer les débits de traitements de messages. Dans [33, 34], les tailles d'images utilisées ne dépassent pas $100\ K$ octets. Dans [58], les messages utilisés ont des tailles inférieures à 500 octets. Dans [39], la taille de fichiers utilisés est inférieure à 100 octets. Dans [110], les données utilisées sont autour de 50205

[3] Des débits entre 15 et $64\ K$bps sont encore utilisés pour des conférences vidéos de moyenne qualité.

octets. Dans [71], la taille de données est en dessous de $300\ K$ octets. Dans [111], la taille des messages ne dépasse pas $3\ K$ octets. Dans [23], la taille de fichiers utilisés est inférieure à $500\ K$ octets. Dans [89], la taille de messages est autour $21.16\ K$ octets.

2. L'étude d'un nombre limité de paramètres de la QdS des SWM. Bien que dans [51] les tailles de données utilisées sont autour de $50\ M$ octets, un seul paramètre de la QdS de SWM est étudié, soit le temps de réponse.

3. L'emploi de méthodes et d'outils moins performants pour la création de SWM. Il existe une multitude d'applications et d'outils fondés sur le protocole SOAP qui s'avère inadéquat pour la création de SWM de haute qualité [39, 97, 99, 127]. Vu que le protocole SOAP a été proposé par de grandes entreprises comme IBM, HP et Compaq, plusieurs outils et logiciels pertinents ont été développés et mis à la disposition des chercheurs et des concepteurs à travers le Web. Autrement dit, le manque de performance de SOAP est compensé en partie par la disposition d'une multitude d'outils et de logiciels connexes qui peuvent faciliter les étapes de conception et de mise en œuvre de SWM fondés sur SOAP.

1.3.2 Objectifs

Pour surmonter les faiblesses 1, 2 et 3 mentionnées ci-dessus, nous avons effectué des travaux d'analyse et de mise en œuvre sur ordinateurs de SWM réels de haute qualité. Nous avons déterminé plusieurs paramètres pertinents de la QdS de SWM réels (comportant des images de tailles allant jusqu'à $4.617\ M$ M octets). Nous avons aussi analysé l'impact de ces paramètres sur les SWM, selon différentes méthodes de création de SWM sur ordinateurs. De plus pour la faiblesse 3, notre objectif est d'effectuer des travaux d'analyse et de mise en œuvre sur ordinateurs de SWM réels de haute qualité. Nous avons déterminé plusieurs paramètres pertinents de la QdS de SWM réels (comportant des images de tailles allant jusqu'à 4.617 M octets). Nous avons aussi analysé l'impact de ces paramètres sur les SWM, selon différentes méthodes de création de SWM sur ordinateurs. Notre objectif est de profiter des caractéristiques de FPGA favorisant une haute QdS à un coût abordable et à une performance supérieure comparativement aux ordinateurs.

Pour faire suite à notre motivation indiquée ci-dessus, nos objectifs de recherche sont les suivants :

- Offrir des méthodes d'analyses de SWM qui sont fondées sur des données réalistes comportant des tailles de paquets diversifiés.
- Déterminer les paramètres les plus importants qui influencent la QdS de SWM. Par la suite, comparer les méthodes existantes de mise en œuvre de SWM.
- Déterminer les meilleures approches qui mènent à la création de SWM de haute QdS sur ordinateurs et sur FPGA.
- Mettre en œuvre des SWM sur différentes plate-formes logicielles et matérielle afin de prouver la pertinence de nos résultats d'analyses.

1.4 Contributions et organisation

1.4.1 Contributions

Les contributions dans cette thèse se résument comme suit :

1. Élaborer des méthodes de conception et de mise en œuvre de SWM sur des plate-formes logicielles hétérogènes, en particulier sur Windows XP, OS X et Solaris. Les méthodes de conception élaborées ont pour but d'offrir une portée étendue et des fonctionnalités variées et de nouveaux SWM tout en gardant une compatibilité maximale entre les plate-formes hétérogènes. Pour atteindre cet objectif, nous étudions les options possibles afin d'offrir des SWM riches en fonctionnalités nouvelles et mis en œuvre sur des plate-formes différentes. Pour ce faire, nous utilisons une interface de création de SWM indépendante de plate-formes. (Voir le chapitre 3).

2. Élaborer une liste de paramètres pertinents qui influencent la QdS selon les résultats obtenus lors de la mise en œuvre de SWM en utilisant le protocole SOAP (voir le chapitre 4) et le style REST (voir le chapitre 5).

3. Élaborer une analyse de performance de SWM basés sur le protocole SOAP (voir le chapitre 4) et sur le style REST (voir le chapitre 5). L'analyse porte sur différents paramètres de QdS ainsi que différentes plate-formes et bandes passantes. Elle comporte l'utilisation de la compression binaire pour trois scénarios : local (à la machine même, appelé LOCAL), sur un réseau local (appelé LAN) et sur le réseau d'Internet (appelé WAN). Deux SWM ont été mis en œuvre avec une procédure spécifique permettant de cueillir les données de performance en temps réel.

L'analyse permet ainsi d'étudier le comportement de SWM réels comportant des images de tailles diversifiées soient entre 1×1 pixel jusqu'à 3822×4819 pixels. Les résultats d'analyse nous ont motivé d'intégrer des méthodes de compression de données binaires telles que DjVu dans les standards du Web de manière similaire à HTTP.

L'analyse est élargie pour couvrir d'autres paramètres de qualité de service liés à la vidéo et au son. (Voir le chapitre 5).

4. Analyser et étudier la corrélation entre la dimension d'une image et le taux de compression, en utilisant des outils comme JPEG, JEPG2000 et DjVu. À souligner que l'originalité de cette contribution est due au fait qu'elle propose une nouvelle piste de recherche qui n'a pas été abordée auparavant par d'autres recherches connexes dans ce domaine. Les résultats de notre étude montrent qu'il est pertinent d'utiliser des images dont le ratio longueur/largeur respecte le ratio d'or, $\frac{1+\sqrt{5}}{2}$, pour augmenter le taux de compression. Ces résultats sont confirmés pour des taux de compression élevés et modérés, c'est-à-dire supérieurs à 0.5. (Voir le chapitre 6).

5. Élaborer une méthode de conception et de mise en œuvre cohérente de SWM sur FPGA en utilisant des outils de conception HDL gratuits, des logiciels de simulation et du code *open-source*. Le choix de mise en œuvre sur FPGA est justifié par les caractéristiques intéressantes des cartes FPGA comme le parallélisme, la vitesse d'exécution, le coût abordable et la faible consommation d'énergie. Notre approche pour la mise en œuvre de SWM sur FPGA favorise l'utilisation unique du HDL (Verilog et VHDL) et l'utilisation de logiciels fournis gratuitement et du code à source ouverte, *open-source*. La mise en œuvre permet d'ajouter des modules plus complexes pour la gestion de la QdS et l'orchestration de SWM. (Voir le chapitre 7).

6. Mettre à jour ainsi qu'adapter du code HDL *open-source* (pour le projet TCP/IP socket qui est utilisé dans la section 7.6.1) et la documentation du module *Ethernet IP Core* qui se trouve dans les annexes A et B, et qui est nécessaire pour la mise en œuvre de SWM sur FPGA dans le chapitre 7, pour la communication entre le FPGA et le port Ethernet sur la carte Nexys3. Ceci a pour effet de faciliter la communication entre FPGA et les périphériques. Ceci a aussi pour effet de faciliter la mise en œuvre de SWM sur la carte Nexys3.

1.4.2 Organisation

Nous avons débuté le présent chapitre 1 en élaborant sur les notions essentielles sur lesquelles se fonde notre travail, telles que les SWM et la QdS dans les SWM (sections 1.1 et 1.2). Nous avons par la suite présenté les motivations et les objectifs de notre thèse (section 1.3). Finalement, nous avons précisé nos contributions (section 1.4).

Dans le chapitre 2, nous présentons l'état de l'art sur la qualité de services dans les services Web. Nous avons ensuite discuté des approches utilisées dans la mise en œuvre de SWM avec contraintes de QdS (section 2.5).

Dans le chapitre 3, nous abordons le sujet de conception et de mise en œuvre de SW sur des plate-formes logicielles hétérogènes ainsi que les options de conception pour garantir une compatibilité et une richesse de fonctionnalités et services. Ce chapitre est un article publié au journal JNW (*Journal of Networks*) [17].

Dans le chapitre 4, les paramètres pertinents de QdS de SWM sont déterminés et un SWM est mis en œuvre selon le protocole SOAP. Une comparaison entre les SWM mis en œuvre selon SOAP et REST est effectuée. Ce chapitre est un article publié au journal JMM (*Journal of Multimedia*) [18].

Le chapitre 5 discute des deux approches logicielles selon SOAP et REST pour la mise en œuvre de SWM. La discussion compare les deux approches selon la liste de paramètres déterminée et selon les mêmes gammes d'images et sur les mêmes plate-formes. Ce chapitre est un article présenté à la conférence MIC-WCMC2011 (*Mosharaka International Conference on Wireless Communications and Mobile Computing*) [20].

Le chapitre 6 présente une contribution propre à cette thèse montrant la corrélation entre les dimensions d'une image et le taux de compression binaire. Une analyse approfondie a été mise en œuvre utilisant trois librairies de compression d'images : JPEG, JPEG2000 et DjVu. Ce chapitre est un article présenté à la conférence MIC-WCMC2011 (*Mosharaka International Conference on Wireless Communications and Mobile Computing*) [19].

Le chapitre 7 traite d'une approche matérielle pour la mise en œuvre de SWM sur FPGA selon le style REST. La discussion compare l'approche matérielle sur FPGA et sur logiciel sur ordinateur. Le chapitre contient aussi un exemple de design de SWM sur FPGA qui a pour but d'offrir une liste de lignes directrices pour aider les concepteurs lors de la mise en œuvre de SWM sur FPGA. Ce chapitre est un article soumis au journal *International Journal of Web and Grid Services*.

La conclusion, les contributions et les travaux futurs de la thèse sont présentés dans le chapitre 8.

Dans l'annexe A se trouvent les spécifications techniques du module *open-source* d'interface avec le Phy utilisé dans la mise en œuvre de SWM sur FPGA.

Dans l'annexe B se trouve la documentation détaillant le fonctionnement du module *open-source* d'interface avec le Phy utilisé dans la mise en œuvre de SWM sur FPGA.

CHAPITRE 2

ÉTAT DE L'ART

2.1 Introduction

Les paramètres de la QdS des SW varient selon le domaine d'application et la nature des données. Par exemple, dans un SW de type transactionnel, les principaux paramètres de QdS sont reliés à la fiabilité des transactions et des données, ce qui favorise l'utilisation du protocole TCP. Comme autre exemple, dans un SW de diffusion vidéo [50], les principaux paramètres de QdS sont reliés aux délais de propagation (comme la gigue) et aux niveaux de congestions de réseau, ce qui favorise l'utilisation du protocole UDP.

La QdS pour Internet peut être classée sous deux catégories :

1. Au niveau de la couche réseau [61, 93] et de la couche transport [11, 64, 90]. Le rôle de la couche transport inclut : le transport de bout en bout, la sélection d'une QdS, la transparence et l'adressage. La couche réseau effectue 3 fonctionnalités principales : le contrôle de flux de données, le routage et l'adressage. Au niveau de la couche transport, les travaux de recherches proposent des améliorations au niveau des protocoles TCP/IP (en anglais, *Transmission Control Protocol/Internet Protocol*) et UDP (*User Datagram Protocol*), les deux protocoles qui résident sur IP. À ce niveau, les recherches sont concentrées sur le contrôle de flux et de paquets. Les paramètres influençant la QdS et qui s'avèrent importants lorsqu'ils dépassent un certain seuil incluent le délai de propagation et le taux d'erreurs de paquets.
2. Au niveau de la couche application [14, 62, 93, 100, 113], les applications adoptent des approches différentes selon leurs plates-formes. Sous cette catégorie, les paramètres qui influencent la QdS incluent la performance, la disponibilité, les délais du traitement CPU et la sécurité. Cette catégorie de QdS est spécifique au domaine de l'application.

Les sujets reliés aux couches réseau et transport sont classés selon trois catégories, notamment selon le flux, le réseau et les protocoles. Dans la section 2.3, nous parlons plus particulièrement de la QdS pour les trafics sur TCP, UDP et les réseaux : virtuels, IMS et mobiles. Nous aborderons ainsi l'amélioration d'algorithmes et des protocoles de la QdS. La QdS au niveau de l'application de SWM est présentée dans la section 2.4.

2.2 Qualité de service dans les couches réseau et transport

Plusieurs paramètres de la QdS s'avèrent importants pour les couches transport et réseau de l'architecture OSI [93, 103] (en anglais, *Open System Interconnection*). Les paramètres de la QdS dans ces couches incluent plusieurs items dont les suivants :

- Le délai d'établissement d'une connexion réseau.
- Le débit de transfert de données.
- Temps de traversé de la connexion.
- Le taux d'erreurs résiduelles.
- La priorité de la connexion réseau.
- Le coût maximal acceptable.

2.3 Qualité de service pour les trafics envoyés sur TCP et UDP

Les communications sur Internet utilisent les protocoles TCP et UDP. TCP est adéquat pour la fiabilité des données, mais il risque d'augmenter les délais lorsqu'il y a des congestions dans le réseau. Par contre, UDP est optimisé pour réduire les délais sans tenir compte des pertes de paquets. UDP est plus adéquat pour les applications et les services multimédia (tels que les diffusions vidéo [50], les conférences multimédia et la téléphone IP) car celles-ci tolèrent des pertes mais ne tolèrent pas les délais. Par contre, TCP est plus adéquat pour les transactions HTTP et le télé-chargement de données FTP, car ces applications tolèrent les délais mais ne tolèrent pas les pertes de données.

Lors d'une congestion, TCP s'adapte selon la bande passante allouée dans le réseau et réduit son débit de transmission de données en conséquence. Par contre, UDP ne réagit pas lors d'une congestion et continue à occuper la même bande passante, ce qui peut aggraver la congestion. Certaines recherches dans le domaine de la QdS essaient d'appliquer des mécanismes qui mélangent des caractéristiques de TCP et UDP car les SW comme d'autres applications sur le Web dépendent des deux types de flots, TCP et UDP [90].

Les contraintes de la QdS pour des services multimédia à des débits variables exigent que la bande passante pour chaque vidéo soit exploitée efficacement. La dulcification

(en anglais, *Smoothing*) [12] est l'une des techniques pour augmenter l'efficacité d'exploitation de la bande passante. Ceci exige que le débit de transmission de la vidéo soit constant le plus longtemps possible. Pour appliquer cette technique, il est souhaitable de rendre égales toutes les trames qui contiennent des images afin d'offrir un taux de trames d'images selon le niveau de QdS souhaité. Une valeur de 25 trames d'images par seconde s'avère adéquate pour plusieurs applications. Cette technique peut être appliquée en-ligne ou hors-ligne pour déterminer deux paramètres : la largeur de la bande passante requise pour la trame dulcifiée et le nombre de trames pour lesquelles la largeur de la bande passante demeure la même. Une demande de vidéo est acceptée ou refusée selon que la bande passante fournie est supérieure ou inférieure à une bande calculée par un algorithme donné.

Une nouvelle méthode [12] en ligne sert à déterminer la largeur de la bande passante pour la vidéo en utilisant les données d'une fenêtre de temps, toutes les largeurs des bandes passantes de toutes les vidéos et la probabilité d'erreur désirée. Les résultats obtenus par simulation et par des scénarios de tests montrent l'avantage de cette méthode pour réduire la bande passante pour une vidéo donnée permettant ainsi de servir un plus grand nombre de clients potentiels.

La stabilité est un paramètre important pour la QdS de la diffusion de vidéo reçue par un client dans un contexte de diffusion multicast à des débits, ou ensemble de couches de débits, variables (en anglais, *layered multicast*) [80, 121]. Dans ce type de diffusions [121], la source a plusieurs clients ayant des accès Internet hétérogènes (vitesses différentes) où la source offre plusieurs versions encodées de la vidéo désirée selon plusieurs débits de données. Le serveur classifie ses clients dans des groupes triés selon leurs débits. Un client peut faire partie d'un groupe à un débit inférieur et un encodage modeste quand sa connexion Internet se dégrade. Par contre, lorsque la connexion Internet s'améliore, le débit augmente et l'encodage peut être de meilleure qualité. Ces changements fréquents de groupe peuvent dégrader la qualité de vidéo visualisée par le client. L'insertion d'une plate-forme d'adaptation, du côté du serveur ou du client, améliore la qualité de la vidéo diffusée. Selon des données sur les débits de chaque client, l'approche adoptée applique un algorithme de sélection pour déterminer le groupe de débit correspondant sans dépasser la limite du nombre maximal des ajouts ou des annulations pour tous les clients dans tous les groupes. Un indice de stabilité doit être minimisé en utilisant une méthode de minimisation. La minimisation se résume en choisissant un vecteur de valeurs optimales (en anglais, *Optimal Rate Vector*) où la

problématique est simplifiée par le fait que le nombre des couches de débits est limité, selon les encodages possibles, pour la diffusion multicast.

Le trafic sur le Web se fonde de plus en plus sur des connexions à courtes durées. Les trames perdues TCP constituent un indice de la congestion du trafic. Par conséquent la source commence à ralentir son débit de trames TCP. Dans [13], une amélioration au protocole de transmissions TCP a été proposée afin d'améliorer la QdS du trafic. La segmentation adoptée est fondée sur le choix de la largeur des trames TCP de façon adaptative, ce qui permet d'utiliser les options suivantes même si la fenêtre de la trame est petite : La retransmission et le redressement rapides. La méthode utilisée est appelée TCP-SF (en anglais, *TCP-Smart Framing*). Dans un contexte TCP traditionnel, l'application envoie une seule trame TCP. En revanche, TCP-SF permet d'envoyer 4 trames ayant la même largeur d'une trame TCP régulier. Ceci permet de recevoir 4 paquets ACKs d'affirmation. Dans ce cas-ci, une perte d'une trame se produit dans l'un des deux cas :

1. Expiration de la durée de retransmission appelée RTO (en anglais, *Retransmission Time Out*), et
2. Réception de 3 copies de ACK déclenchant l'algorithme de retransmission appelé FR (*Fast Retransmission*).

Ceci permet à la source de savoir au plus tôt ce qui se passe du côté récepteur, à savoir s'il y a une congestion ou non. L'idée est d'envoyer des petites trames à chaque cycle lent de transmission, notamment au début de flot. Lors d'une congestion, la source s'ajuste en réduisant les tailles des paquets TCP (segmentation), sinon la taille des trames revient à celle d'un TCP régulier, sans segmentation imposée.

L'avantage de TCP-SF est qu'elle n'influence pas les algorithmes classiques du protocole TCP : Le début lent (en anglais, *Slow Start*), l'évitement de congestion (en anglais, *Congestion Avoidance*), la retransmission rapide (en anglais, *Fast Retransmission*) et le redressement rapide (en anglais, *Fast Recovery*). L'inconvénient de cette méthode est que l'information ajoutée pour la gestion de l'en-tête augmente 4 fois plus comparativement à TCP régulier. La méthode peut utiliser une technique de segmentation non homogène pour réduire le nombre de segments dans le cas où il n'y a pas de congestion.

Le trafic sur Internet contient plus de vidéos diffusées comme les films, la télévision et les télé-conférences. Le protocole UDP est mieux adapté que TCP pour ce domaine puisque les diffusions vidéos sont peu vulnérables aux pertes de paquets, mais plus

sensibles au délai en raison de la propagation sur réseau. Les mécanismes de retransmission TCP offrent plus de fiabilité et plus de délais. Les paquets TCP ont des tailles différentes, un débit varié, selon l'état de congestion du réseau, pourtant les paquets vidéo ont normalement des tailles égales et un débit constant. Dans le domaine de la diffusion vidéo sur Internet, l'article [26] propose un nouveau protocole, appelé VTP (en anglais *Video Transport Protocol*), de diffusion vidéo qui tient compte de la gestion de la congestion afin de ne pas influencer la qualité de la vidéo encodée selon MPEG-4. L'aspect nouveau de ce protocole proposé est le fait qu'il permet d'estimer la largeur de la bande passante disponible pour le récepteur, sachant que les vidéos diffusées sur Internet exploitent le protocole UDP qui n'a pas de mécanisme d'adaptation lors d'une congestion.

Les logiciels populaires comme *RealPlayer* et *Media player* de *Windows* offrent des services pour les vidéos en temps réel sur UDP avec des capacités limitées, une petite taille de fenêtres vidéo et une résolution modeste. Lors d'une congestion, VTP [26] s'adapte en changeant l'encodage de la vidéo. Dans ce dessein, un serveur garde plusieurs versions de la vidéo à diffuser selon plusieurs encodages différents. Lors d'une congestion, le serveur sélectionne une version allégée pour la diffusion, c'est-à-dire que l'encodage devient plus léger et que les tailles des paquets vidéo deviennent plus minces. Ceci garantit un débit de trames constant au récepteur. VTP a été ainsi conçu pour être utilisé de bout à bout (en anglais, *end-to-end*), c'est-à-dire qu'il ne dépend pas des techniques de la QdS dans les routeurs, de RED (en anglais, *Random Early Drop*), d'AQM (en anglais, *Active Queue Management*) et d'ECN (en anglais, *Explicit Congestion Notification*). VTP est fondé sur UDP et il utilise un mécanisme de contrôle de la congestion au niveau de l'application. VTP est concentré sur la rétroaction entre le récepteur et le serveur où le premier envoie des ACK au serveur qui les utilise afin d'estimer le temps d'aller-retour et la congestion. Le serveur est donc en mesure de choisir la meilleure version de vidéo à diffuser, c'est-à-dire celle ayant l'encodage le plus adéquat. Normalement, l'augmentation du trafic UDP, qui est constitué principalement d'images, du son et de la vidéo, peut réduire la bande passante disponible pour le trafic TCP.

L'une des méthodes proposées permettant aux trafics TCP et UDP de coexister de façon équitable [116] propose des techniques pour augmenter la QdS des transferts vidéo. Les paramètres de la QdS pertinents dans ce contexte incluent la stabilité et l'extensibilité FGS (en anglais, *Fine Granular Scalability*) de façon à ne pas trop pénaliser le trafic

TCP. Le FGS offre un algorithme de compression adéquat pour la diffusion de la vidéo qui est déjà intégré dans le standard d'encodage MPEG-4.

Les applications comme HTTP, FTP et TELNET exploitent TCP afin de transférer les données sans perte. TCP est fondé sur des mécanismes de retransmissions et de contrôle de flots de données dans des fenêtres (trames) de transmissions. Lors de congestion, TCP s'adapte et réduit son débit de transmission. Par contre, UDP ne réagit pas en cas de congestion. Certains articles proposent un règlement de débit appelé TFRC (en anglais, *TCP-Friendly Rate Control*) afin de partager de façon égale la bande passante entre TCP et les autres protocoles comme UDP, dans le cas où les deux types de trafic (TCP et autres) suivent les mêmes trajectoires. L'article applique plusieurs techniques comme celle de FEC (en anglais, *Forward Error Correction*) pour des flots de vidéo encodée selon MPEG-4. La technique employée par le FEC élimine les erreurs dans les paquets multimédia dues aux pertes. Le serveur envoie un nombre spécifique *f* de paquets redondants pour un certain nombre de paquets réguliers. Le récepteur utilise ces paquets redondants seulement en cas de pertes pour reconstruire les paquets manqués. Le mécanisme reste valide si le nombre de paquets manqués est moins que *f*. Lors d'une congestion, il est pertinent d'utiliser moins de paquets redondants en choisissant une valeur plus petite de *f*.

Une technique de gestion AQM (en anglais, *Active Queue Management*) est proposée pour améliorer la QdS [92]. AQM permet aux routeurs d'exploiter des messages d'informations appelés indices de la source (en anglais, *Source Hint Messages*). Ces messages sont placés dans l'entête du paquet IP comme des indices sources (*source hints*). Dans ce dessein, l'entête IP désigne entre 4 et 17 octets pour offrir des services aux applications qui exigent un certain niveau de QdS selon deux paramètres simultanément : le délai et le débit supportés par le réseau. À noter que l'augmentation du premier paramètre risque de réduire le deuxième. Pour cette raison, l'application doit choisir ces paramètres soigneusement.

L'un des mécanismes d'amélioration de l'équité (en anglais, *Fairness*) entre des protocoles multiples est proposé pour le réseau *DiffServ* [59] (en anglais, *Differentiated Services*). Il s'agit d'un mécanisme de transport qui est fondé sur les librairies du logiciel TCAPI (en anglais, *Traffic Control API*). Il a été proposé à l'IETF dans le domaine de la QdS pour IP. Cette équité doit exister entre les flots basés sur TCP et d'autres protocoles comme UDP. Les flots UDP ne s'adaptent pas lors d'une congestion, ils peuvent ainsi congestionner le trafic sur le réseau disponible défavorisant par conséquent les flots TCP.

2.3.1 QdS des réseaux virtuels, IMS (IP Multimedia Subsystem) et mobiles

Un client mobile possède normalement un seul appareil qui exploite plusieurs technologies sans fils en sélectionnant une configuration selon ces technologies. Une solution pertinente pour améliorer la performance de TCP [11] se résume en changeant la taille maximale des segments TCP et en modifiant des algorithmes TCP lors d'un changement d'une technologie à une autre. Le protocole *IP mobile* constitue une source supplémentaire de délais pour les réseaux sans fils. Par exemple, quand un client voyage ayant son appareil cellulaire, sa connexion change d'une cellule mobile à une autre (en anglais, *Handover*). Lors d'une connexion TCP, ceci peut interrompre temporairement l'échange de données avec le réseau central. Ceci peut être mal interprété par TCP qui considère cela comme une congestion de réseau et par conséquent, il réduit son débit de données transmises. Les informations du trajet de connexion TCP changent lors d'un changement de technologie par l'appareil cellulaire. Ces informations incluent la largeur de bande passante disponible, le temps de propagation aller-retour et le seuil de retransmission (en anglais, RTO : *Retransmission Time out*).

Pour les réseaux virtuels fondés sur des circuits ou bien des lignes dédiées virtuelles (en anglais, *Virtual Leased Lines*), un mécanisme d'adaptation est proposé [94] pour modifier les trajets ou chemins (en anglais, *paths*) dans le dessein de respecter les paramètres déterministes de la QdS. Normalement, ces réseaux virtuels relient des branches d'une entreprise ou de deux institutions différentes. Le mécanisme engendre un contrôleur fondé sur la logique floue pour ajuster les ressources allouées à un trajet selon les variations du trafic dans ce trajet, dans le but de respecter les contraintes de la QdS. Un délai acceptable du trafic est l'un des paramètres pertinents de la QdS à améliorer en supposant que les informations du lien virtuel sont fournies à l'avance par un protocole de routage étendu, par exemple.

Considérons SIP (en anglais, *Session Initiation Protocol*) [21] qui est un protocole de signalisation destiné à l'établissement des sessions de communications pour plusieurs applications dont la téléphonie sur Internet. Une amélioration à SIP a été proposée par l'article [52] afin d'améliorer la QdS dans les réseaux IP, en particulier les réseaux exploitant *Diffserv*. L'approche dans cet article permet à SIP de gérer des informations sur la QdS. Elle exploite COPS (en anglais, *Common Open Policy Service*) qui est un protocole permettant aux serveurs de politiques d'envoyer leurs décisions aux équipements du réseaux comme les routeurs, pour le contrôle de l'admission dans le réseau *Diffserv*.

Dans ce réseau capable de maintenir la QdS, les entités COPS interagissent avec des serveurs SIP (appelé Q-SIP) et avec des médiateurs de largeur de bande passante BB (en anglais, *Bandwidth Broker*). Un client de COPS (celui-ci est effectivement le *Proxy* Q-SIP comme expliqué dans les lignes qui suivent) demande une réservation de ressources auprès d'une entité COPS appelée COPS-DRA (*COPS-DiffServ Resources Allocation*) qui se trouve dans le routeur[1] situé aux bornes du réseau *Diffserv*. Quand un appelant SIP lance une requête INVITE, celle-ci est interceptée par un serveur Proxy Q-SIP. Ce dernier décide si la session doit tenir compte de la QdS ou non. Si oui, il ajoute des paramètres de la QdS à INVITE et l'envoie au prochain serveur Q-SIP jusqu'au destinataire. Quand ce dernier accepte l'appel, son Q-SIP Proxy lance une demande de réservation de ressources auprès du serveur central de l'admission des ressources BB.

L'architecture que définit le 3GPP pour un système multimédia sur IP, appelé IMS (en anglais, *IP Multimedia Subsystem*), inclut les éléments principaux suivants [67] :

- Un protocole de signalisation (SIP).
- Des architectures fournissant la QdS pour le réseau IP, notamment *InteServ* (en anglais, *Integrated Services*), visant les données en paquets, et *DiffServ*, pour les données en flots. *InteServ* est un autre standard dans le même domaine. *DiffServ* est plus extensible (en anglais, *scalable*) mais moins dynamique comparativement à *InteServ*. MPLS (en anglais, *Multi Protocol Label Switching*) est une technologie qui peut jouer un rôle important dans ce domaine, comme un *back-bone* pour *Diffserv*. L'utilisation d'*InteServ* est plus coûteuse en gestion puisqu'il exploite RSVP (en anglais, *Resource ReServation Protocol*) pour chaque flot de paquets.
- Un contrôle des politiques du réseau (en anglais, *PCF : Policy Control Function*). Ces entités sont reliées, entre autres, à l'authenticité, à l'autorisation, à la gestion des ressources et à l'admission des appels. COPS est un candidat pour l'interaction entre les entités qui décident des paramètres de la QdS et des PEPs (en anglais, *Policy Enforcement points*, un routeur en est un exemple) qui les exécutent. Ces entités décideurs sont appelées PDPs (en anglais, *Policy Decision Points*), par exemple, un BB *Bandwidth Broker*.

L'article [67] analyse aussi la coopération entre les protocoles de l'IMS. L'analyse effectuée peut servir dans un contexte d'un fournisseur de service dans un réseau IP.

Parmi les options technologiques offertes pour renforcer la QdS, il y a en particulier ATM (en anglais, *Asynchronous Transfer Mode*), RSVP, *InteServ* et *DiffServ*. L'article [31]

[1] Dans l'article, l'entité du routeur est implanté dans un ordinateur exploitant la plate-forme Linux. Pour simplifier l'architecture de plus, l'entité du BB réside aussi dans cet ordinateur. Le routeur joue le rôle de fournisseur de la QdS.

s'attarde sur l'architecture de *DiffServ* pour la QdS pour IP, puisqu'elle est plus extensible (en anglais, *scalable*) que *InteServ*. L'article propose une nouvelle approche pour garantir la QdS de bout à bout, permettant aux clients d'accéder à des services où la QdS est gérée de façon automatique (en anglais, *Plug and Play*). L'approche se résume en développant un outil pour classifier en temps réel les différents trafics dans un réseau. L'avantage de cet outil est qu'il peut être adapté afin de gérer dynamiquement la largeur de bande passante libre dans les réseaux *DiffServ*, en particulier, pour servir comme une entité supplémentaire du médiateur de la bande passante (en anglais, BB, *Bandwidth Broker*). L'approche est fondée sur PHB (*Per Hop Behaviour*) où PHB fait la distinction entre les paquets en les marquant selon leurs types en utilisant le code de 6 bits appelé DSCP (en anglais, *Differntiated Service Code Point*). Ce code DSCP est placé dans un champ réservé pour *DiffServ* dans l'en-tête IP.

Pour accommoder le trafic multimédia sur les réseaux IP, l'IETF et l'ITU ont adopté le protocole RTP/RTCP (en anglais, *Real Time Protocol/RealTime Control Protocol*) [25]. Pour la diffusion multimédia en temps réel, on a adopté une version spécifique de RTP appelée RTSP (en anglais, *Real Time Streaming Protocol*). RTP est normalement exploité par les applications, vu qu'il est au-dessus du protocole UDP. L'une des façons d'éviter le délai pour le trafic de la voix, la taille des paquets est amincie pour réduire la probabilité de fragmentation et d'égarement dans les routeurs. Ainsi, du côté client, l'application ne subit pas de dégradations à cause de pertes de paquets et de congestions.

Considérons le protocole RSVP qui permet de réserver des ressources sur le réseau. Des modifications ont été apportées à RSVP dans [56] pour garantir un niveau acceptable de la QdS afin de surmonter des inconvénients comme le manque d'extensibilité et sa complexité. Les auteurs de l'article [56] proposent des améliorations pour rendre RSVP plus rapide et pour y ajouter de nouveaux mécanismes dont la mobilité, ce qui peut servir l'IP mobile. Les protocoles reliés à la QdS sur IP se classent sous deux catégories :

1. ceux utilisant une approche par paquet et
2. ceux utilisant une approche par flot comme le protocole RSVP.

Pour la première catégorie, les paquets contiennent des informations de la QdS sans intervention de la part des routeurs. Pour la deuxième catégorie, les ressources sont réservées d'avance en utilisant des messages spécifiques entre la source de données et sa destination (client). Cette approche sépare la signalisation et les données, ce qui favorise l'extensibilité.

2.4 Qualité de services Web multimédia au niveau de l'application

L'emploi d'un intermédiaire ou d'un courtier est l'une des méthodes valides pour améliorer la QdS selon [123]. Ce courtier (en anglais, *Broker*) peut être indépendant ou bien inclus du côté client ou du côté serveur afin de permettre au client de choisir un niveau de QdS pour un service donné. Par conséquent, le courtier négocie des paramètres de la QdS avec des serveurs qui fournissent le service désiré. Le courtier utilise des algorithmes spécifiques pour la maintenance et la gestion de la QdS, notamment :

- HQ (*Homogeneous Resource Allocation Algorithm*) assigne un niveau de QdS homogène à tous les clients, donc le service est ajusté selon le nombre de clients. Pour accommoder un nouveau client, le serveur doit rajuster la QdS à tous les clients. Quand les ressources sont abondantes de nouveau, le serveur augmente la QdS à tous les clients.
- RQ (*Non-Homogeneous Resource Allocation Algorithm*) assigne un niveau varié de QdS aux clients selon leurs besoins. Pour accommoder un nouveau client, le serveur rajuste la QdS à certains clients s'il le faut. Lorsque les ressources sont abondantes, le serveur n'augmente pas la QdS aux clients, par contre il réserve des ressources libres pour des clients virtuels, potentiels ou futurs.

L'architecture proposée dans [123] influence légèrement le temps d'exécution d'un service Web car la négociation de la QdS se fait lors de la découverte d'un service Web et avant son exécution. Le courtier de la QdS est constitué des trois composants suivants :

1. Un gestionnaire d'informations de la QdS : Il permet de recueillir des informations sur la QdS auprès des UDDIs afin de trouver des serveurs ou fournisseurs de SW répondant aux exigences de la QdS souhaitée. Par la suite, le gestionnaire contacte les serveurs et sauvegarde ses informations dans une base de données qui sera mise à jour régulièrement. La sélection d'un serveur de service est faite selon l'information dans cette base de données.
2. Un négociateur de la QdS : Une fois la sélection d'un serveur faite, le négociateur contacte le serveur directement afin de confirmer l'information de la QdS. Si la négociation échoue, le négociateur essaie de nouveau un autre serveur parmi une liste de candidats.
3. Un analyseur de la QdS : Il sert à évaluer les services d'après les expériences vécues des clients qui envoient des comptes-rendus donnant des informations

sur le délai des réponses du serveur et de la fiabilité de service. Ces informations aident à choisir un meilleur service ou serveur potentiel.

L'article [123] propose d'inclure l'entité du courtier de la QdS dans le côté client, serveur ou bien de façon indépendante. Pour une meilleure sélection d'un SW et pour une meilleure gestion de la QdS, nous croyons que le courtier doit être dans le module de l'UDDI et du fournisseur, comme dans [127] qui propose une entité située dans l'UDDI pour effectuer des tâches d'analyseur de la QdS.

Pour les SW multimédia, le flot de données est partagé entre les données multimédia et le contrôle. La QdS est l'un des éléments importants qui valorisent un service multimédia donné. À cet effet, l'article [126] propose des améliorations aux protocoles SOAP et CC/PP (en anglais, *Composite Capability/Preference Profiles*).

Dans [128], la découverte des SW tenant compte de la QdS est effectuée à l'aide d'une nouvelle ontologie complémentaire à DAML-S (en anglais, *DARPA Agent Markup Language-Service*) appelée *DAML-QoS ontology*.

L'article [125] aborde le test et la vérification de la QdS des SW, en particulier pour des SW combinés. Les logiciels existants pour tester la QdS des applications logicielles doivent êtres adaptés à la nature dynamique et répartie des SW.

2.5 Approches de mise œuvre de services Web multimédia

Dans les sections précédentes, nous avons discuté des paramètres de la qualité de services et des mises en œuvre de SWM selon les deux architectures SOA (représentée par SOAP) et ROA (représentée par REST). Dans cette section, nous abordons le sujet de mise en œuvre de SWM selon les outils et les plate-formes. Les méthodes de mises en œuvre de SWM se résument comme suit :

- Méthodes logicielles, en particulier sur ordinateur et les GPU (en anglais, *Graphical Processing Unit*) [36, 49].
- Méthodes matérielles, en particulier sur ASICs (en anglais, *Application Specific Integrated Circuits*) et FPGA (en anglais, *Field Programmable Gate Arrays*).

Voici quelques points de comparaison :

- Les GPU et les ordinateurs ont des modèles de traitement fixes, alors que les FPGA peuvent être adaptés selon un modèle variable de traitement.
- Les GPU offrent un moyen de traitement logiciel accéléré et adapté pour faire des calculs arithmétiques avec des chiffres de point flottant. Par contre, le FPGA offre un moyen d'accélération de traitement plus flexible pouvant être branché à des périphériques et divers systèmes.
- L'utilisation de GPU nécessite d'avoir un ordinateur vu que les GPU se branchent normalement sur un bus standard de données PCIe dans l'ordinateur. Les GPU offrent une vitesse de traitement et un débit élevé sans garantir l'aspect temps réel. À l'inverse, un FPGA peut garantir des délais très étroits sans pouvoir offrir un débit de traitement très élevé.

Les caractéristiques des GPU se résument comme suit :

- Ils ont un modèle de traitement fixe et adapté pour le traitement rapide de données et des algorithmes mathématiques avec point flottant.
- Ils sont exploités depuis un ordinateur puisqu'ils possèdent un seul moyen de connexion avec le bus de données d'ordinateur. Pour ce faire, il est nécessaire d'utiliser un langage de programmation évolué comme C.
- Ils consomment beaucoup d'énergie comme les CPU comparativement aux FPGA et ASICs.
- L'emploi des GPU ne nécessite pas d'expertise sur le matériel et l'architecture interne d'ordinateur ou des GPU.
- Le coût de GPU est abordable.

Vu que les GPU ont beaucoup de similarités avec les ordinateurs, le sujet de GPU ne sera pas abordé dans les prochaines sections.

Les ASICs offrent une meilleure performance, une taille et des dimensions réduites et une meilleure économie énergétique, comparativement aux ordinateurs, aux GPU et aux FPGA. Les inconvénients des ASICs se résument comme suit :

- Coût élevé.
- Manque de flexibilité et difficulté d'effectuer des modifications importantes ou des mises à jour sur un système déjà fabriqué.
- Temps relativement long entre la conception et la mise sur le marché.
- Nécessité une étape de prototype, de simulation et de test avant la production qui est typiquement accomplie par des FPGA[2] ou des ordinateurs.

[2]Les FPGAs offrent une plate-forme intéressante pour faires de prototypes pour les ASICs, vu le temps relativement court entre la conception et la production.

- Nécessite d'outils spécialisés et complexes.

En raison des inconvénients ci-dessus des ASICs, nous avons écarté cette solution pour la mise en œuvre de SWM. Nous considérons donc uniquement les mises en œuvre sur ordinateur ou sur FPGA.

2.5.1 Mise en œuvre de SWM sur ordinateur

Les ordinateurs furent le moyen préféré pour la mise en œuvre de SWM et des systèmes informatisés. Les avantages de cette méthode peuvent être résumés comme suit :

- Une grande gamme d'outils et de langages de programmation (comme C et Java) qui sont compatibles sur plusieurs systèmes d'exploitations (comme OS X, Linux, Unix et Windows) et architectures de processeurs (comme ARM et Intel).
- Une performance acceptable pour la plupart des domaines d'applications, en particulier pour les SWM.
- Populaire et abordable.
- Plusieurs applications.

2.5.2 Mise en œuvre de SWM sur FPGA

Les FPGA [97] offre des caractéristiques intéressantes pour la mise en œuvre de SWM et des systèmes informatisés. Les avantages de cette méthode peut être résumé comme suit :

- Un seul langage de programmation à maîtriser soit HDL (en anglais *Hardware Description Language*, comme VHDL et Verilog).
- La possibilité d'intégration avec d'autres outils de création qui permettent de générer du code HDL (du VHDL et du Verilog) depuis des codes en C et depuis des MEF (Machine à États Finis).
- Une performance supérieure comparativement à l'ordinateur pour la plupart des domaines d'applications, en particulier pour les SWM.
- Un coût abordable.
- Des dimensions physiques minimes comparativement à un ordinateur portable, par exemple.
- Une consommation faible d'énergie comparativement à un ordinateur et à un GPU.
- Du parallélisme pour des fonctions et processus multiples.

- Une haute performance. Dans certaines applications comme le traitement en temps réel sur des fenêtres glissantes d'images, la performance offertes par des FPGA dépassent celles des ordinateurs et des GPU [49].
- Une multitude de méthodes de branchement standard sur des bus de données comme PCIe dans un ordinateur ou sur des ports d'entrées-sorties polyvalents.

Les FPGA offrent un compromis et un équilibre entre la haute performance, la petite taille et l'économie énergétique.

Dans chapitre 7, nous présentons les détails de la mise en œuvre de SWM sur FPGA en utilisant :

- La carte Nexys3 qui contient, entre autres, la puce FPGA Spartan6 avec des périphériques diverses comme un port Ethernet à un débit de 10/100 Mbps, un afficheur à sept segments, des buttons poussoirs, des interrupteurs, des ports entrée-sortie, et un port USB.
- Des outils de conception et de programmation sur FPGA gratuits.
- Du code HDL, notamment du Verilog et du VHDL, et des logiciels open-source pour la simulation.

CHAPITRE 3

LES SERVICES MULTIMÉDIA ET LES SERVICES WEB

Avant-propos

Auteurs et affiliation :

AL-CANAAN Amer ; étudiant au doctorat et chargé de cours au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

KHOUMSI Ahmed ; professeur au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

Date d'acceptation :

30 novembre 2009.

État de l'acceptation :

version finale publiée.

Revue :

Journal of Networks (JNW).

Référence :

A. Al-Canaan and A. Khoumsi. «Cross-platform Approach to Advanced IP-Telephony Services using JAIN-SIP ». Journal of Networks, vol. 5(7), pp. 808-814, July 2010.

Titre français :

Une approche sur des plate-formes multiples pour des services de téléphonie IP avancés utilisant JAIN-SIP

Contribution au document :

Cet article correspond à la contribution 1 indiquée à la section 1.4. Cette contribution consiste à élaborer des méthodes de conception et de mise en œuvre de SWM sur des plate-formes logicielles hétérogènes, en particulier sur Windows XP, OS X et Solaris. Les méthodes de conception élaborées ont pour but d'offrir une portée étendue et des fonctionnalités variées et de nouveaux SWM, tout en gardant une compatibilité maximale entre les plate-formes hétérogènes. Pour atteindre cet objectif, nous étudions les options possibles afin d'offrir des SWM riches en fonctionnalités nouvelles et mis en œuvre sur des plate-formes différentes. Pour ce faire, nous utilisons une interface de création de SWM indépendante des plate-formes. (Voir le chapitre 3).

Résumé français :

Ce chapitre introduit des méthodes de création de services multimédia sur des plate-formes hétérogènes en visant l'objectif de garantir une compatibilité maximale et de pouvoir élargir la portée et la diversité des services réalisés.

SIP (en anglais, *Session Initiation Protocol*) est un protocole de contrôle et de signalisation qui a été adopté par le 3GPP pour fournir des services multimédia destinés aux réseaux de télécommunications mobiles. La tâche de créer des services téléphoniques IP à travers des plate-formes multiples s'avère ardue. Malgré l'émergence d'outils tel que JAIN-SIP qui facilite le développement de services téléphoniques IP indépendants de plate-formes, le développement de services SIP avancés comportant la vidéo sur des différentes plate-formes reste encore plus difficile. Nous discutons de quelques solutions qui permettront d'étendre JAIN-SIP afin d'améliorer et de supporter la création de services comportant la vidéo sur des plate-formes multiples.

Cet article fait la démonstration de la conception et de la mise en œuvre de quelques services avancés en utilisant JAIN-SIP dans un environnement de plate-formes multiples, incluant Windows, OS X et Solaris. En même temps, nous montrons des détails sur

notre solution proposée et proposons l'utilisation de services Web (SW) pour multiplier les fonctionnalités offertes sur JAIN-SIP permettant d'améliorer la création de services multimédia sur des plate-formes multiples en utilisant l'outil JAIN-SIP. L'article présente des détails et des options de conception incluant les services Web (**SW**) et démontre la création et la mise en œuvre de quelques services avancés, en sur des plate-formes variées comme Windows, OS X et Solaris. D'autres services simples et intermédiaires sont aussi montrés pour expliquer la pertinence de notre approche.

Mots clé : services téléphoniques IP sur des plate-formes multiples, services Web (SW), SIP, JAIN-SIP, QuickTime, JMF, FMJ, JNI, Qualité de services (QdS).

3.1 Abstract

SIP is a call control and signalling protocol that was adopted by the 3GPP[1] to deliver IP multimedia services to the mobile network. Cross platform IP-telephony service creation is a challenging task. Although, the emergence of JAIN-SIP[2] has reinforced the development and implementation of platform-independent IP-telephony services, the development of advanced cross-platform SIP (Session Initiation Protocol) services including video-enabled services is more challenging. Several solutions are discussed in order to extend the ability of JAIN-SIP to enhance advanced and video-enabled service creation on multiple platforms. This paper demonstrates the design and implementation of a few advanced services using JAIN-SIP in cross-platform environment including Windows, OS X and Solaris, providing more details on our proposed solution and proposing the use of Web services (**WS**) to bring in more features to JAIN-SIP. Some simple and intermediate services are also presented for explanation of our approach.

Index terms— Cross-platform advanced IP-telephony services, Web services (WS), SIP, JAIN-SIP, QuickTime, JMF, FMJ, JNI, Quality of service (QoS).

[1] *3rd Generation Partnership Project*.
[2] *Java Application interface for Integrated Networks using SIP*.

3.2 Introduction

SIP [101] is a call control and signalling protocol that was adopted by the 3GPP to deliver IP multimedia services to the mobile network. Cross-platform IP-telephony service creation is a challenging task. Being a Java-based API (*Application Programming Interface*) for SIP, JAIN-SIP has reinforced the development and implementation of platform-independent IP-telephony services. Yet the development of advanced cross-platform SIP services including video-enabled services is more challenging. Several solutions are proposed in order to extend the ability of JAIN-SIP to enhance advanced and video-enabled service creation on multiple platforms. In this paper we demonstrate our cross-platform design solution and implementation of a few advanced services using JAIN-SIP. Some simple services are also presented for explanation of our design approach. The implementations and validation were carried out on three platforms: OS X, Windows XP and Solaris.

We emphasise also the challenges and the practical aspects we faced during our design and implementations.

This paper is divided into four distinct sections. The first section highlights some related works and provides an overview of SIP and JAIN-SIP. The second section presents several solutions to cross-platform implementation. The third section deals with our SIP services design. The fourth section presents our implementation and validation of the designed services. The fifth section presents our conclusion and future work.

3.3 Related work

As an extension to our previous work in [15], this paper presents more details on our proposed solution and proposes the use of Web services (**WS**) [54] to bring in more features to JAIN-SIP. JAIN-SIP [15, 37, 43, 104, 129] enhances the creation of IP-telephony services at the SIP protocol layer level. The interoperability between SIP and H.323[3] is achieved by the use of Web services [124]. The latter approach is probably more suited when two different signalling protocols are involved.

[3]H.323 is another mature VoIP (*Voice-over-IP*) protocol that was developed by the ITU-T (*International Telecommunication Union Standardisation Sector*).

In [24], an architecture is proposed to manage complex multimedia services using JAIN-SIP and CPL (*Call Processing Language*). In [79, 107], the mobility features of SIP are applied in Ad-Hoc networks using JAIN-SIP. In [47], an architecture for policy personification using CPL and JAIN-SIP is provided. Some of the technologies available for NGN (*Next Generation Networks*) to achieve flexible and easy service creation are mentioned in [73].

The theme of this paper is to extend the JAIN-SIP API to include multiple platforms where there are some restrictions on the use of video capture features as will be explained later on. However, we believe that advanced cross-platform IP-telephony service creation has hardly been addressed. Most research was focused on the design and implementation of simple or intermediate services that do not include video, which was intended mostly for a single platform.

3.4 Overview of JAIN-SIP

JAIN-SIP is a Java-based API that we have chosen to implement our cross-platform advanced services. It provides a stack, which handles low-level message processing and production at the SIP protocol layer. The architecture of JAIN-SIP is based upon a consumer/producer event model, where a SIP message is either consumed or produced according to a specific event.

To implement SIP services using JAIN-SIP, an application layer has to be built. We have built our application layer upon the *Sip Communicator* [105] project, which is an open source Java application implementation of JAIN-SIP. It provides basic and intermediate IP-telephony services including voice and text messaging.

3.5 Proposed solution

The advanced services in the following sections rely on video capture feature as well as the text messaging and HTML (*Hyper Text Markup Language*) content. The latter are intermediate features.

Our design approach is cross-platform and thus our goal is to implement video-enabled services on multiple platforms, including OS X, Windows XP and Solaris.

The video capture feature, upon which our video-enabled services are based, is supported by JMF (*Java Media Framework*) for only Windows and Solaris platforms, but not for OS X. Java does not support audio and video by default. To add multimedia support to Java applications, *Sun Microsystems* provides JMF as an optional package that has to be installed separately for several target platforms. Java and JMF packages support cross-platform applications programming merely for voice-based IP-telephony services.

Our solution to this dilemma was to find alternatives to JMF that can either replace it or work with it. Several options are presented in the following sections.

3.5.1 Pure Java solutions

Pure Java solutions means that only JDK (*Java Development Kit*) and Java-based API are considered, which include the following:

1. **the use of a hybrid formula incorporating QTJ** (*QuickTime for Java*), which is an API and application framework providing support for audio, video and images [60]. QTJ is available only for Windows and OS X. Our design approach [15] enabled the following scenario: when communicating between Windows and OS X, video is handled, i.e. captured and compressed using **QTJ**, while JMF is used between Windows and Solaris.

2. **the use of open-source code such as FMJ** (*Freedom for Media in Java*) [48], which may provide another hybrid solution. **FMJ** is compatible with JMF and is supported on three platforms: Windows, OS X and Linux. Since this option and option 1 are alike, we demonstrated the application of option 1 only.

3.5.2 Non-pure Java solutions

This means that languages other than Java are considered such as C/C++ and perl.

1. **the use of JNI** (*Java Native Interface*) [102]. **JNI** can be used to wrap efficient C/C++ code and make it accessible through Java. With **JNI**, it is possible to call C/C++ code from Java programs and vice versa. Although this option brings in more performance due to native code utilisation, it needs higher programming skills compared to Java, since it involves platform-specific programming.

2. **the use of WS** (Web services). Although it was proposed in [124] as a solution to the interoperability between SIP and H.323 protocols for VoIP applications on different machines and locations, it can offer a reliable solution when used to enhance cross-platform advanced services using JAIN-SIP (see Figure 3.1). In this case, the performance and thus QoS would be even better, since the propagation delay of **WS** [130] messages[4] through the network would be eliminated.

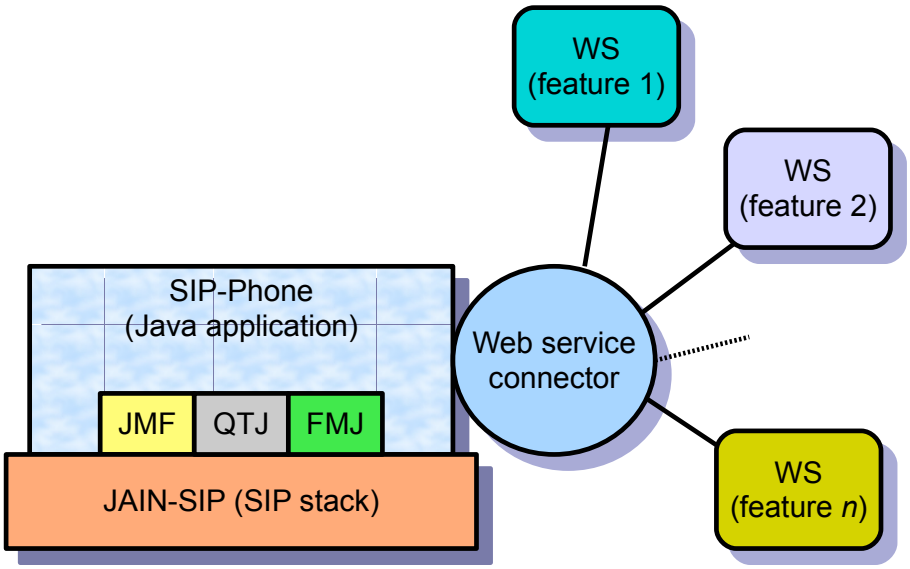


Figure 3.1 The use of **WS** to extend JAIN-SIP applications with new features.

In this paper, results of the **QTJ** solution are presented on Figure 3.1, while the other solutions are left for future work.

QTJ and **FMJ** are pure Java solutions, while **JNI** and **WS** are API and platform-independent. However, **JNI** is limited to C/C++, i.e. integrating Java with C/C++ code. Table 3.1 summarises the four options mentioned above.

The **JMF** is the default Java multimedia package in JAIN-SIP and its video capture feature is supported only on Solaris and Windows. **WS** are not limited to Java, which means that video capture feature or any desired feature can be implemented by C++ or by some scripting language.

With regard to compatibility with **JMF**, **FMJ** is the only candidate since it is intended to be an alternative to **JMF** and be compatible with it by having similar methods.

[4]Whether the **WS** is based upon SOAP (*Simple Object Access Protocol*) standard or the simple RESTful [130] style.

Table 3.1 Comparing the four proposed solutions.

Property	Solution			
	QTJ	**FMJ**	**JNI**	**WS**
Purely Java	Yes	Yes	No	No
JMF compatible	No	Yes	No	No
Platforms supported	2	3	many	many
Performance	better	good	better	variable

QTJ supports 2 platforms (Mac OS X and Windows), while **FMJ** supports 3 platforms (Mac OS X, Windows and Linux). On the other hand, **JNI** and **WS** support unlimited number of platforms.

Finally, the same score of performance is given to **QTJ** and **JNI** since they incorporate native code directly, which is done implicitly with **QTJ**. Although **WS** can be built using Java, they can offer better performance when implemented using efficient API and native code. Interestingly, **WS** features integrated with Web service connector (as shown on Figure 3.1) would outperform regular **WS** over distributed environments (Web), since the former are meant to serve and integrate with JAIN-SIP locally, for instance, on the same machine bringing new features and sub-services to it. A thorough performance comparison is left to future work.

3.6 SIP Services conception

We have designed several IP-telephony services, including advanced video-enabled and intermediate services, using the JAIN-SIP as our SIP protocol layer. In this section, we present the conception procedure, as shown on Figure 3.2, and the scenarios of four designed and implemented services. Since SIP servers are not mandatory in SIP architecture, though they can provide additional support and services, we present each service by using a SIP message flow diagram between UAs without any SIP server. The implementations and the user interfaces are presented in section 3.7.

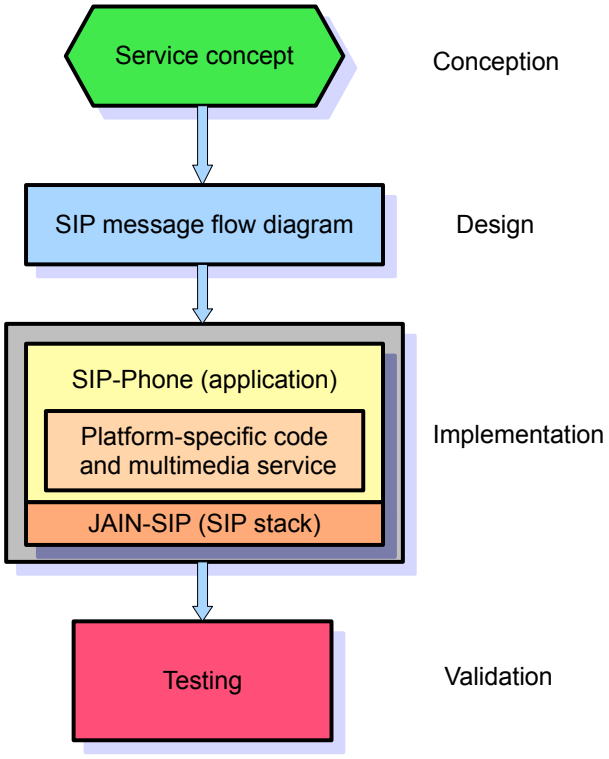


Figure 3.2 SIP services conception procedure.

3.6.1 Service 1: Auto Answer (AA)

This service enables the user to receive calls automatically without his intervention, i.e. without picking up. This can be handy for hands-free mode or for call centres as well as for other domains where calls are answered immediately, one after another.

The message flow diagram of this service is presented on Figure 3.3, where the **AA** is activated for user $\mathcal{B}$. As soon as an *INVITE* request message is received (step 1) at the UA of $\mathcal{B}$ ($UA_{\mathcal{B}}$), *TRYING* and *RINGING* response messages are sent back to $\mathcal{A}$ (steps 2-3). Since the call is accepted automatically, a *positive final response* (*200-OK*) is sent back (step 4), which is acknowledged (step 5) and the media session is initiated (step 6). When the call is ended a *BYE* request message is sent (step 7). Finally an *200-OK* confirms the disconnection request and thus the media session is stopped.

3.6.2 Service 2: Instant Messages (IM)

This service is accomplished by the use of the *MESSAGE* request. Our goal was to make this service available as an independent service as well as during a call session.

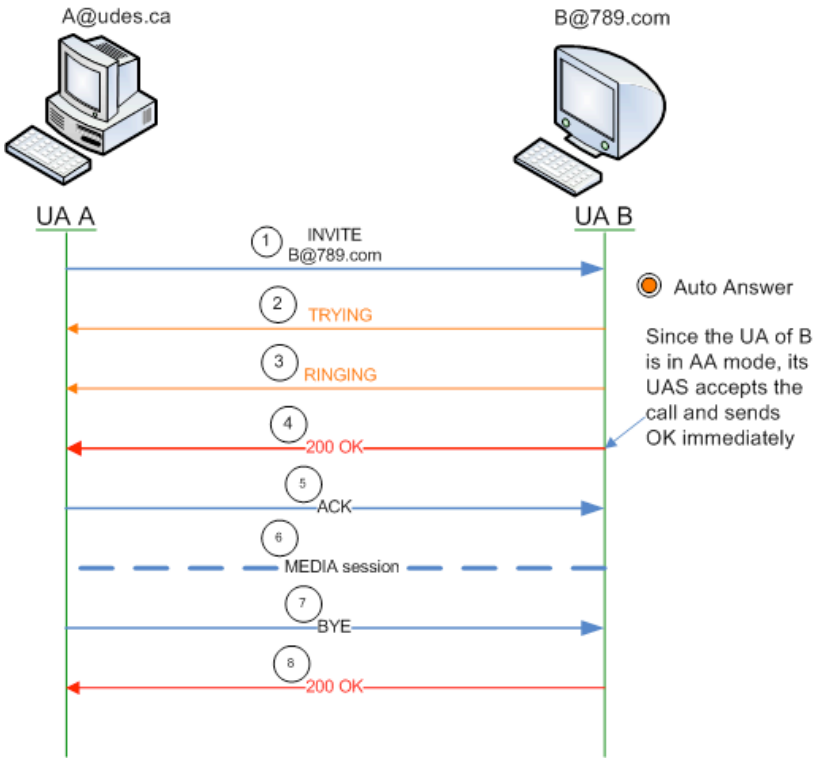


Figure 3.3 Message flow diagram illustrating the *Auto Answer* (**AA**) service.

The diagram on Figure 3.4 illustrates the message flow of the **IM** service between two UAs. At the application layer, user $\mathcal{A}$ types a message for $\mathcal{B}$, thus the UA of $\mathcal{A}$ ($UA_{\mathcal{A}}$) sends a *MESSAGE* request to $UA_{\mathcal{B}}$ (step 1). The message is processed and displayed. Then, $UA_{\mathcal{B}}$ sends an *OK* response back to $UA_{\mathcal{A}}$ (step 2). The same scenario is repeated over from the opposite direction (steps 3 and 4).

3.6.3 Service 3: Advanced Offline and Busy Messages (AOBM)

With respect to our design criteria, the **IM** service may be utilised even during a call session. To benefit from this feature during a call or when a user is busy, we have developed an Advanced Offline and Busy Messages service (**AOBM**). When the **AOBM** service is activated, the following options are offered to the the calling person:

1. view the called person's Web page as HTML content and obtain any relevant information that might be available,

2. leave an offline text message, which can be stored in an offline message box, and

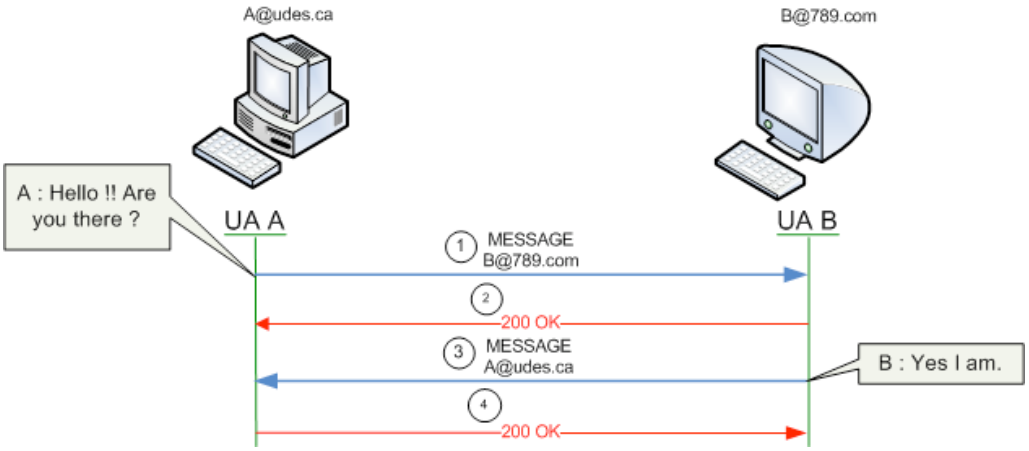


Figure 3.4 Message flow diagram illustrating the *Instant Message* (**IM**) service.

3. leave an offline video message using a web cam, which can be played back when the called person becomes available.

The message flow diagram of the **AOBM** service is shown on Figure 3.5, where a radio button activates the service for user $\mathcal{B}$ (running Mac OS X).

An *INVITE* request is sent to the $UA_{\mathcal{B}}$ (step 1). Response messages *TRYING* and *RINGING* are sent back to $UA_{\mathcal{A}}$ (steps 2-3). Since user $\mathcal{B}$ is busy, a *486-BUSY HERE* response is sent directly to $UA_{\mathcal{A}}$ (step 4). $UA_{\mathcal{A}}$ acknowledges the *486-BUSY HERE* response by sending an *ACK* request to $UA_{\mathcal{B}}$ (step 5).

Then, an automatic HTML *away message* is sent by $UA_{\mathcal{B}}$ and accepted by $UA_{\mathcal{A}}$ (steps 6-7). Once user $\mathcal{A}$ (running Windows XP) reads the away message, he wishes to reply back by typing another offline text message (step 8), to which $UA_{\mathcal{B}}$ responds by an *OK* (step 9).

When $\mathcal{A}$ decides to send an offline video message using his webcam, $UA_{\mathcal{A}}$ informs $UA_{\mathcal{B}}$ (step 10) that an offline video message follows shortly, using a special *MESSAGE* request, to which $UA_{\mathcal{B}}$ replies by sending a *200-OK* response (step 11). A special TCP (*Transmission Control Protocol*) client class becomes active in order to receive the video. When $\mathcal{A}$ is done with his video message, a special TCP server class sends it to $UA_{\mathcal{B}}$ (step 12). The received video is saved locally to be viewed when $\mathcal{B}$ becomes available. For clarity, some illustrations are put directly on Figure 3.5.

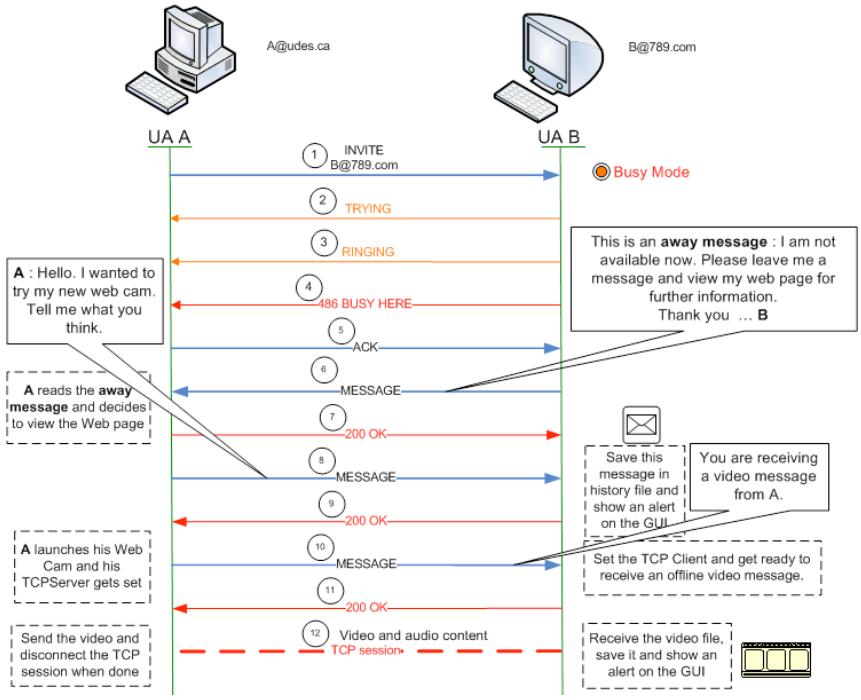


Figure 3.5 Message flow diagram illustrating the *Advanced Offline and Busy Messages* (**AOBM**) service.

3.6.4 Service 4: Video Calling using webcam (VCW)

This service enables video streaming during call sessions. For simplicity, the intended scenario for this service is that a webcam can be activated and controlled manually during a call as illustrated in the diagram of Figure 3.6. Steps 1-6 represent a regular established audio session.

When user $\mathcal{A}$ activates his webcam during a call, a *MESSAGE* request is sent by $\text{UA}_{\mathcal{A}}$ to $\text{UA}_{\mathcal{B}}$ (step 7), and then a video session is initiated between $\mathcal{A}$ and $\mathcal{B}$ (step 8). Steps 9-10 correspond to call hang-up requested by $\mathcal{A}$.

We have implemented this service utilising **QTJ** for OS X and Windows, RTP (*Real Time Protocol*) and M-JPEG (*Motion-JPEG*) video format. This simplified method converts video frames into sampled and compressed JPEG images. The voice is transmitted by default using JMF [82].

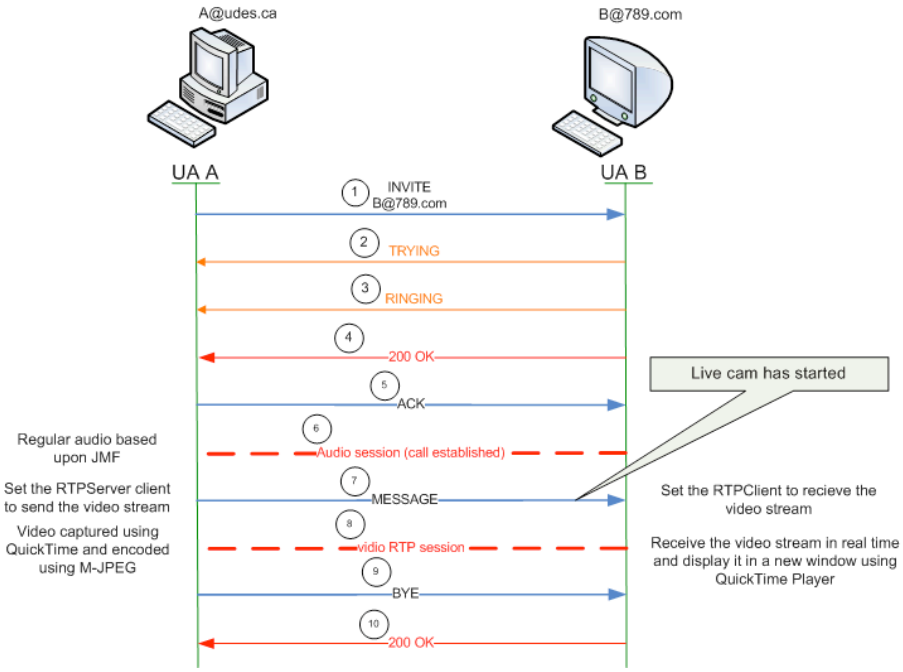


Figure 3.6 Message flow diagram for the *Video Calling using Web Cam* (**VCW**) service.

3.7 Implementation and validation

In this section we present our four implemented and validated services at the application layer, using our *SIP-Phone* application. We present briefly the main GUI (*Graphical User Interface*) of our *SIP-Phone* as described on Figure 3.7.

- **AA service**; this service is activated using a radio button on the main GUI of the *SIP-Phone* application. An action listener associated to this radio button sets a static variable, indicating whether the **AA** service is active or not. Some changes have been made to the classes that process the *INVITE* request in order to send the *TRYING* and *RINGING* responses according to the message flow diagram of the service.

- **IM service**; the rectangular button **message/video**, as shown on Figure 3.7, is used to activate the **IM** service directly when a valid SIP address is entered. A special GUI, similar to that used for the **AOBM** service as shown on Figure 3.8 with fewer options, includes a JTextPane and a JEditorPane components in order to display both of the received message and the typed message, respectively. We have added other features to this GUI, such as font and colour selection.

Figure 3.7 The main GUI for our SIP-Phone.

- **AOBM service**; the radio button **Busy Mode** and the **message/video** button are used to activate the service and to send an offline message, respectively. A similar GUI of the **IM** service with added options is used to show the offline messages, text and HTML, involved in this service as shown on Figure 3.8.

  A visual notification appears on the main GUI as shown on Figure 3.7 to indicate the presence of offline messages (text and video) in the in-box.

  The video is captured using web cam and is encoded to reduce its size. The transmission of video is accomplished by using TCP sockets, such that the offline video message is downloaded without interfering with any active call session. The service deals with several video formats such as the *QuickTime* ***mov*** or ***mpeg***. A special GUI using *QuickTime* handles the playback of video messages.

- **VCW service**; this service can be activated during a call session or during an **IM** session, where a button is used to launch the web cam (see Figure 3.8). Several video formats can be used for video streaming. For test purposes, the selected video streaming format used was M-JPEG (*Motion-JPEG*), which is transmitted using the RTP (*Real Time Protocol*).

Figure 3.8 A GUI of the **AOBM** service showing an offline HTML message.

3.7.1 Test and validation

The implemented advanced services were tested and validated at both the protocol layer and at the application layer, mainly on OS X and Windows XP. An essential SIP diagnostic and trace tool (see Figure 3.9) was designed and built within the SIP-Phone so as to verify the message flow diagrams and to collect performance data about messages. The trace log is used to construct the message flow diagram for each service. The direction of message flow is shown as well as the complete SIP message with headers and SDP content, where applicable. We also have tested the simple and intermediate services on Solaris, Windows and OS X.

3.8 Conclusion

An important contribution of this paper is that it presents the main guidelines for SIP-service design and implementation of multimedia services in cross-platform environments. We have addressed the main challenges facing the designer of advanced IP-telephony services when using JAIN-SIP, whether seeking a pure Java implementation or seeking performance. We provided a hybrid solution, based upon **QTJ**, for cross-platform advanced IP-telephony services design and implementation for OS X, Windows XP and Solaris. Our approach can be extended to include other platforms such as Linux. We have discussed the possibility of three other solutions; namely, the use of open-source code such as **FMJ**, the use of **JNI** and the use of Web services.

Figure 3.9 Trace log for the *Auto Answer* (**AA**) service between Mac OS X and Solaris.

The limitations inherent in JAIN-SIP are mostly related to Java. The performance can be an issue for JAIN-SIP as well as for Java in general, but at the same time it provides a user-friendly layer for cross-platform design.

The changes needed to accommodate and cope with each platform at the Java code level were few. Cross-platform design and implementation is submitted to several factors such as codec's and platform-specific features that should be carefully considered.

3.8.1 Future work

We would like also to extend our approach to include Linux in order to investigate another hybrid solution, based upon **FMJ** and **JNI**, for multimedia-based services using Java and JAIN-SIP. We would consider the application of Web services combined with JAIN-SIP in order to provide more features that are not feasible by Java and that can be built better and more efficiently using native API or non-Java API. Other related domains of interest include QoS, security and feature interactions amongst Web services and advanced cross-platform SIP services.

CHAPITRE 4

QUALITÉ DE SERVICES WEB MULTIMÉDIA

Avant-propos

Auteurs et affiliation :

AL-CANAAN Amer ; étudiant au doctorat et chargé de cours au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

KHOUMSI Ahmed ; professeur au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

Date d'acceptation :

6 juin 2011.

État de l'acceptation :

version finale publiée.

Revue :

Journal of Multimedia (JMM).

Référence :

A. Al-Canaan and A. Khoumsi, «Multimedia Web Services Performance : Analysis and Quantification of Binary Data Compression ». Journal of Multimedia (JMM), Vol 6, No 5 (2011).

Titre français :

La performance de services Web multimédia : une analyse et une quantification de compression de données binaires

Contribution au document :

Cet article correspond aux contributions 2 et 3 indiquées à la section 1.4. Les contributions de cet article se résument comme suit :

- Élaborer une liste de paramètres pertinents qui influencent la QdS selon les résultats obtenus lors de la mise en œuvre de SWM en utilisant le protocole SOAP.
- Élaborer une analyse de performance de SWM basés sur le protocole SOAP. L'analyse porte sur différents paramètres de QdS ainsi que différentes plate-formes et bandes passantes. Elle comporte l'utilisation de la compression binaire pour trois scénarios : local (à la machine même, appelé LOCAL), sur un réseau local (appelé LAN) et sur le réseau d'Internet (appelé WAN). Deux SWM ont été mis en œuvre avec une procédure spécifique permettant de cueillir les données de performance en temps réel.

  L'analyse permet ainsi d'étudier le comportement de SWM réels comportant des images de tailles diversifiées soient entre 1×1 pixel jusqu'à 3822×4819 pixels. L'analyse propose d'intégrer des méthodes de compression de données binaires telles que DjVu dans les standards du Web de manière similaire à HTTP.

Résumé français :

Les services Web multimédia constituent une charge considérable sur la largeur de la bande passante d'Internet et sur les serveurs Web. La demande croissante de services Web multimédia fait en sorte qu'un des principaux soucis des fournisseurs de services Web est de maintenir un niveau adéquat de qualité de service.

La compression binaire de données, avec ou sans perte, augmente le gain en performance et maintient ainsi une bonne qualité de service lors des mises en œuvre à petite ou grande échelle. Au détriment d'un temps de traitement de CPU plus élevé, la compression de données permet de maintenir la stabilité de paramètres de la qualité de

service tels que le temps de réponse, le temps de télé-chargement et le délai de serveur dans un contexte d'une bande passante limitée.

Cet article présente une analyse détaillée d'un service Web multimédia créé selon SOAP et basée sur une plage de données représentative et réaliste, afin de quantifier le gain en performance de services Web multimédia dû à l'utilisation de la compression binaire d'image. Pour offrir des données en entrée à notre méthode analyse, un service Web multimédia de récupération d'images a été mis en œuvre comportant des ensembles d'images sous deux formes comprimées et non comprimées.

Le travail mené dans cet article encourage l'utilisation de compression de données multimédia et propose l'intégration de techniques de compression multimédia dans les standards actuels de services Web multimédia comme dans HTTP.

La méthode de l'analyse ainsi que les résultats obtenus peuvent être élargis et adaptés pour inclure divers services et architectures de réseaux tels que les réseaux mobiles locaux, les réseaux Ad-Hoc, les réseaux mobiles, les réseaux de la téléphonie IP et les systèmes répartis.

Mots clés : services Web, performance, Qualité de service, compression d'image, temps de réponse, délai de serveur, bande passante.

Note :

Une version préliminaire et moins élaborée a été publiée dans [16].

4.1 Abstract

Multimedia Web services constitute a considerable load on Internet bandwidth and on Web servers. With the increasing demand of multimedia Web services, the need to maintain a respectable QoS is a major concern for multimedia Web services' providers. Binary data compression, lossy and lossless, increases performance gain and maintains QoS, at large and small-scale implementations. At the expense of increased CPU processing time, data compression provides a reliable solution to sustain QoS parameters such as response time, download time and server delay where bandwidth is limited.

This paper presents a detailed analysis, which is based upon realistic and representative data, in order to quantify the performance gain for multimedia Web services due to the application of binary image compression. To provide data for our analysis method, we have built an image-retrieval Web service that provides a set of images in both compressed and non-compressed form. The work in this paper promotes the utilisation of multimedia compression techniques and proposes the integration of multimedia compression techniques in current Web standards such as HTTP.

The analysis method and results can be extended and adapted to various service and network models, such as wireless LAN, Ad-Hoc, mobile networks, IP-telephony and distributed systems.

Index terms— Web services, performance, QoS, image compression, response time, server delay, bandwidth.

4.2 Introduction

Web services are capable of conveying text and binary data. The exchange of multimedia including images, voice and video, is very common amongst Web users. This phenomenon adds more pressure on the resources of the Web, including available bandwidth. Thus, the best-effort Web can hardly cope with the increasing demand on multimedia services while sustaining acceptable quality of service (QoS).

To help accommodate new multimedia Web services, while improving the QoS, two natural solutions that can be considered include the following [72, 98]:

- Improve the medium [30, 106], by increasing the bandwidth. This may necessitate hardware upgrade and new infrastructure, which may or may not be feasible due to lack of investment and high cost. For example, when replacing a $56 - Kbit/s$ modem by a $512 - Kbit/s$ DSL (*Digital Subscriber Line*) or cable modem.

- Improve the media [30, 77, 118], by decreasing the message size through applying data compression, for example. This seems to be more effective, since it requires typically software upgrade and no change on the existing costly bandwidth and infrastructure.

Generally, for a multimedia Web service, payload is binary encoded and constitutes the major part of the sent data. Therefore we focus our attention on the payload itself, rather than on the header information, which can be text or XML-encoded[1].

Binary data compression (whether lossy or lossless) improves QoS for both small and large-scale implementations where bandwidth is limited. The improvement of QoS is not only valuable to Web services that are based upon SOA[2] and ROA[3] [130], but also to other types of traffic on the Web including FTP (*File Transfer Protocol*) and IP-Telephony.

Although the utilisation of binary data compression reduces several QoS parameters such as response time, server delay and download time, the central processing unit (*CPU*) takes additional time during compression and decompression. The CPU contributes to the local factors that affect the parameters of the QoS. On the other hand, the response time is susceptible to different factors that make it uncertain. However, the extra processing time can be minimised by the use of optimised software code, more CPU and memory resources.

This paper presents a thorough and quantitative analysis of the performance obtained when applying binary compression to multimedia Web services. For the purpose of our analysis, we have developed an image-retrieval Web service, which provides various measurements and calculations of relevant QoS parameters. The open source DjVu [45] compression library is utilised in order to compress, decompress and display the image samples. The various measurements involve LAN (*Local Area Network*), WAN (*Wide Area Network*) and locally within one computer (localhost) configurations, with different hardware resources, bandwidth and platforms.

The analysis method and results can be extended, adapted and applied to improve several QoS [109] parameters in various types of networks, such as wireless LAN (WLAN) [76], Ad Hoc[4], mobile networks, IP-Telephony [15] and distributed systems.

The contribution of this paper include: 1) the methodology of the analysis including the detailed discussion of its results, which quantify accurately the performance improvement for the implemented multimedia Web service based upon realistic and representative

[1] XML, eXtensible Markup Language.

[2] *Service Oriented Architecture*, which is based upon the SOAP (*Simple Object Access Protocol*) protocol.

[3] *Resource Oriented Architecture*, which is based upon the RESTful (*Representational State Transfer*) style.

[4] Ad Hoc, refers to networks that do not require a router or a wireless access point to establish a connection between two or more computers.

image samples, 2) the proposition to standardise image compression tools and formats into HTTP upon which Web services are built.

This paper is divided into six distinct sections. Section 4.3 highlights some related works. Section 4.4 gives an overview of data and image compression. Section 4.5 presents the development of an image-retrieval multimedia Web service and Web client application utilising an efficient library of image compression, i.e. DjVu. This Web service provides the input data for realising our analysis. Section 4.6 presents the performance analysis method, various service configurations and provides a discussion of the results. Finally, section 4.7 consists of the conclusion and future work.

4.3 Related work

A comparison between two lossless compression techniques, namely, RLE and Huffman encoding [86], is presented in [8]. An implementation of a multimedia Web service is provided in order to demonstrate the compression ratio and the compression speed for both encoding techniques.

The performances of Web services, ASP (*Active Server Pages*) and RMI (*Remote Method Invocation*) [39, 71, 111] are compared showing that the data overhead of Web services' messages is non-negligible, which lowers the throughput and increases the response time of Web servers. The throughput of Web servers is measured with and without payload compression. The performance of Web services can outperform that of RMI when document-oriented style of communication is utilised, rather than RPC (*Remote Procedure Call*). Generally, RMI is faster than SOAP Web services and thus some work has been done to determine when it is more adequate to utilise RMI rather than SOAP, according to predetermined QoS parameters such as availability and accessibility. Other factors that speed up SOAP applications include certain of XML-processing tools [68].

QoS contract composition is part of Web services orchestration and choreography. The work in [96] proposes a probabilistic approach to QoS soft contract composition that is based upon probability distributions rather than fixed figures. The objective is to produce more natural and less pessimistic QoS contracts that can be generated using Monte-Carlo simulations.

Reducing the message size through compression [30, 65, 87, 109] enhances the security and the QoS for mobile devices running XML-based Web services. The security is enhanced through XML data encryption. The utilisation of XML compression and then encryption reduces the transmission time as well as the CPU processing time. Compression reduces message size and thus leaves less data to be processed by the encryption algorithm. Obviously, this approach is beneficial when the payload itself is composed mainly of XML data. Binary XML data compression technique results in a trade-off between response time and throughput. Data compression reduces messages sizes and thus increases throughput. On the other hand, the response time increases due to compression and decompression overhead [30]. The security level, which includes authentication, encryption and encapsulation of data, has an impact on the performance of wireless networks. Different levels of security, up to ten levels, were tested on different configurations with several performance parameters such as mean response time and throughput.

Other proposed approaches to improving the performance of Web services at the transport layer include HTTP caching and UDP (*User Datagram Protocol*) message binding [113, 119]. UDP offers fewer headers, low message overhead, no error correction and connection-less behaviour. Binding protocols such as HTTP (*Hypertext Transfer Protocol*), FTP and SMTP (*Simple Mail Transfer Protocol*) are compared with UDP in terms of performance. On the other hand, caching web responses can be beneficial for read-only Web services or when the responses do not vary in a relatively short time.

Our work can be distinguished from those mentioned above in several aspects. Firstly, we concentrate our research on lossy compression in multimedia Web services, where the payload data are in binary format (images, voice and video) rather than XML. Secondly, we study the influence of binary image compression on Web services both at the client and at the server sides in order to analyse several relevant QoS parameters such as response time, download time and processing time. Thirdly, we cover up a wider range of data sizes, which is more realistic and common to multimedia Web services.

Compared to our work in [16], we provide more details about our on-line analysis method including the pseudo code. We provide also more analysis and discussion about the three scenarios (LOCAL, LAN and WAN) including the experimental results of the response time.

4.4 Data and image compression

Payload sizes of multimedia Web services can be considerably large. When dealing with high-resolution images and high quality video, data sizes can be above $1\ MB$, which increase packet congestion probability, lower QoS and augment bandwidth overload. Two solutions come to rescue the situation, namely, increasing the bandwidth and utilising data compression.

The first solution may involve new hardware upgrade and new infrastructure, which may not be feasible without adding more investment cost. We believe that this solution is effective merely in the short run, since it would eventually lead to saturation and degradation of QoS. On the other hand, our tests amongst many others reveal clearly a gap between the actual and the nominal bandwidth figures [72]. For example, for an Internet cable modem with nominal download rate of $512\ Kbit/s$, the actual download rate was about $75\ Kbit/s$.

The second solution is utilising data compression. Different data compression techniques can be utilised efficiently to help Web services keep up with the growing demand on multimedia Web contents and respect their QoS constraints. Several approaches propose compressing SOAP headers. This is less effective for multimedia Web services, since the length of SOAP headers is much smaller than the payload size.

Lossless data compression is inherent in HTTP 1.1 standard and is available for Web browsers and Web clients. The supported formats include those generated by *gzip* (LZ77), *compress* (LZW) and *deflate* (zlib) [83, 86]. This has considerable influence on Web services, which are based upon XML and text content. On the other hand, multimedia Web services require adequate compression tools (lossy and lossless) are to be adopted and standardised. The DjVu library is a promising tool, due to its high image compression ratio, which can be utilised in conjunction with the existing compression standards in order to achieve full compression to textual content and binary payload.

The application of a high compression ratio utility is desired so as to maintain the available Web bandwidth and QoS for multimedia Web services as well as for other types of services such as IP-telephony. The choice of a data compression technique is based upon the following criteria [85]:

- Compression ratio.
- Speed of compression and decompression.

- Compatibility and ease of conversion between different formats.

Usually, binary compression libraries utilise one or more algorithms of compression to achieve higher compression ratios. The open source DjVu library [45, 57] relies on the classification of every pixel of an image into either foreground or background. The foreground includes text or drawings, while the background includes pictures, paper textures and colours. The foreground is compressed using pattern-matching technique, while the wavelet technique [66] is utilised for the background. For example, JPEG images can be compressed and displayed efficiently as DjVu format, denoted djvu. However, other equivalent image compression tools can be utilised.

The djvu-image format cannot be converted directly to JPEG (*Joint Photographic Experts Group*) format. An intermediate step has to take place, i.e. converting from djvu to TIFF (*Tagged Image File Format*). Other tools and libraries can be utilised to convert between various image formats. For example, ImageMagick [74] is another utility, which is an open source project that is available for a variety of platforms including UNIX, OS X and Windows.

For demonstration purposes, the main characteristics of two tools of DejaVu, namely *c44* and *bzz*, are compared to the characteristics of PPM [85][5], with three orders, namely PPM(4), PPM(6) and PPM(16) as represented on figure 4.1. The compression ratio ($bits/Byte$) for a given image is calculated as its compressed size in bits divided by its original size in bytes (the lower the better). The speed of compression and decompression are expressed as KB/s (the higher the better). The *c44* image compression tool is based upon the wavelet technology (suited for only image compression), while the *bzz* and the PPM encoding utilities are general purpose since they can be used to compress text and binary image data. According to our test results as shown on figure 4.1, PPM(6) offers the highest compression and decompression speeds, while *c44* has the best compression ratio ($3.2\ bits/Byte$). As the order of PPM is increased, the compression and decompression speeds become lower due to the increasing number of iterations of the PPM algorithm. Figure 4.1 shows that the decompression of the *bzz* tool is more sensitive to machine processing capabilities compared to compression (the test was held with two different machines: OS X having more resources, in terms of CPU and memory, and Linux with fewer resources).

[5]Prediction by Partial Matching (PPM). The order of the PPM technique as in $PPM(n)$ refers to is the number of the previous symbols such that $0 \leq n \leq 16$.

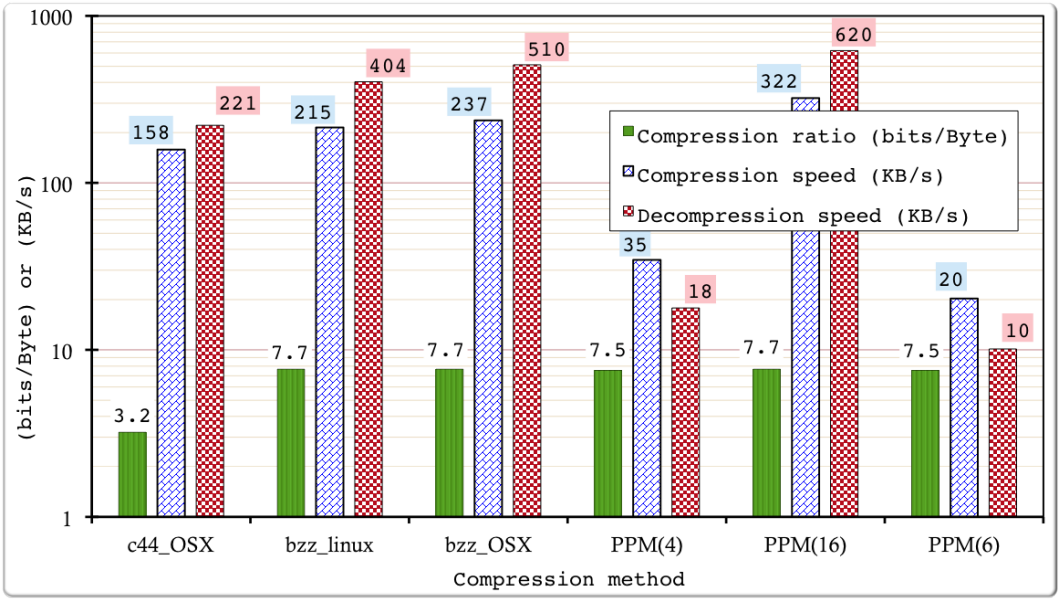


Figure 4.1 Comparison of some image compression tools on different platforms.

4.5 Image-retrieval multimedia Web service development

To help quantify the influence of data compression on multimedia Web services, we developed an image-retrieval Web service and a Web client application using NetBeans 6.5 IDE (*Integrated Development Environment*) and GlassFish server v2.1. A good tutorial for Web service implementation with NetBeans can be found in [88]. This section provides details about the image-retrieval Web service and the Web client we developed. The Web client issues requests, which are processed at the Web service, and collects relevant data for analysis.

4.5.1 Development environment

For comparison, each image is available in its original JPEG format as well as in djvu-compressed format (compressed in advance using the *c44* tool of the DjVu library because it offers a respectable compression ratio).

Our developed Web service involves several standard API's and tools [44], namely:

- **JAX-WS** (*Java API for XML-Based Web Services*): it is part of Java EE and it enables the creation of RESTful as well as SOAP Web services and is intended to replace JAX-RPC. The current version utilised in our implementation is 2.1.

- **JAXB** (*Java Architecture for XML Binding*): it is used to bind Java objects to XML schemas and vice versa. This makes it possible to process XML as Java objects and to represent Java objects as XML.

- **WSIT** (*Web Services Interoperability Technology*): an extension to JAX-WS intended to enhance security and interoperability with Microsoft's Windows Communication Foundation.

- **MTOM** (*Message Transmission Optimisation Mechanism*): a mechanism to send binary data, as raw bytes, separately from the SOAP data. One way to include binary data inside SOAP is to utilise base-64 encoding (using 64 printable characters, thus each character is encoded using six bits). To encode three bytes (twenty-four bits) of binary data utilising base-64, it takes four bytes, since the 24 bits are taken six bits at a time to produce four characters. Obviously, this is inefficient since the size of binary data gets larger, due to the base-64 encoding, which is necessary in order to fit within a SOAP message.

In addition to the Web service itself, we developed a comprehensive Web client agent that measures various QoS parameters. The message flow diagram between the Web client and Web service is shown in figure 4.2. The Web server responds to client requests, which includes getting an image by name, querying its size, returning a list of all available image names, and calculating the time duration needed to process the request and prepare the required image bytes to be sent back to the client.

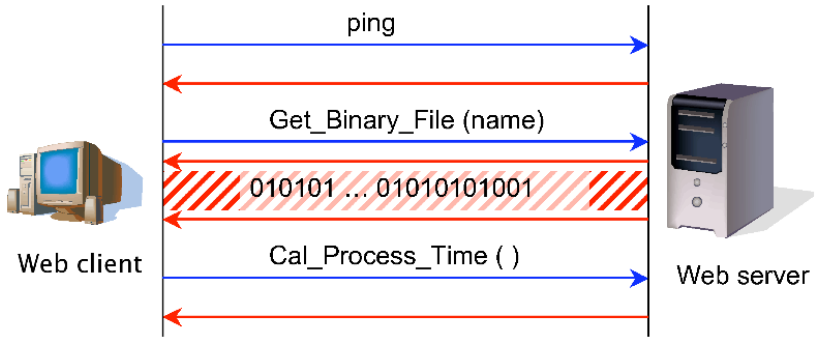


Figure 4.2 Overview of the image-retrieval Web service.

At the Web server side, three entities can be distinguished, namely, a Web application, a Web service and a Java bean. At the outer level, the Web application wraps both the Web service and the Java bean, while the Web service wraps the Java bean. The requests are delegated down to the Java bean, which acts roughly as a database. Since the internal structure of the Web server side components is invisible to the client, we refer to them by the abstract term *Web service*.

The Web client application has two roles:

1. Retrieving and displaying a set of 20 images on a specially designed GUI (*Graphical User Interface*, not shown here).
2. Measuring and saving several parameters for further analysis, such as the following (to be explained more in sections 4.6.2 and 4.6.1):
   - **ping_time**, which is the time needed for a tiny packet to travel from one host a to another host b and back to its origin, i.e. host a.
   - **processing delay** time at the server (***S_Delay***), which is the time needed for a server to process a request. This time is between receiving a request at the server and sending a response.
   - **response time**, which is the time duration between sending a request at the client and receiving a response.
   - the actual **download time**, which is the time between receiving the first and last bits of data.
   - **image-rendering time**, which is the time between receiving the last byte of an image and displaying the image on the GUI.

4.5.2 Network configuration

Our Web service and Web client were developed for several platforms (see table 4.1) that are located at two different domains:

- Domain 1, with nominal download speed of $512\ KB/s$ for WAN and $10\ MB/s$ for the LAN, which consists of Mac OS X and Linux operating systems.
- Domain 2, with nominal download speed of $10\ MB/s$ for WAN and $10\ MB/s$ for the LAN, which consists of Windows XP, Linux and Sun Solaris operating systems.

The implementations are built with the most recent Java versions for each platform, i.e. jdk1.5 on OS X (G5) and jdk1.6 elsewhere.

Table 4.1 System and platform configurations.

Domain	Platform	RAM, CPU@clock frequency
1	OS X 10.4	1.25 GB, G5@1.6 GHz
	Linux 2.6	386 MB, PIII@0.7 GHz
2	Windows XP	512 MB, PIV@2.5 GHz
	Windows XP	2 GB, Intel D@2.8 GHz
	Linux 2.6	2 GB, Intel D@2.8 GHz
	Solaris	512 MB, UltraSPARC@0.4 GHz

Although we demonstrate our analysis method on the wired-network, we believe that the same approach can be extended to cover the wireless networks [76] with the following generalised model:

- Bit error rate. Although, the wireless network has much higher bit error rate due to the radio link, the wired network has a very low bit error rate.
- Link loss probability. This factor is higher for wireless networks. On the other hand, wired networks can suffer from link loss due to different causes such as weather conditions (corrosion of cables), hardware failure of routers and power failure.
- Mobility and roaming. Mobility for wired networks is limited compared to the wireless network.

4.6 Performance Analysis

In this section, we analyse the performance of the multimedia Web service we have developed. The Web service provides ten images in JPEG format and in djvu format by utilising the *c44* tool [45], which provides a respectable compression ratio as shown on figure 4.1 (a total of twenty images). The selected images sample span actual sizes from 1×1 pixel up to 3822×4819 pixels, corresponding to 655 bytes up to $4617.087\ KB$.

The ten JPEG images were selected arbitrarily from various sources, and thus they are compressed differently.

The analysis is realised for the following scenarios:

1. **LOCAL**, where the client and server code share the same resources of a single machine (local host). This configuration enables rapid verification and preliminary testing before deploying the actual implementation on the Web, since the delay times and response times are lower. This configuration provides guidelines in order to validate the calculated results.

2. **LAN**, where the client and server are on two different machines within the same local area network (LAN). Two different LANs were available to our tests: fast LAN in domain 2 with $10\ Mbit/s$ download speed and slower LAN in domain 1 with nominal download speed of $0.512\ Mbit/s$. This configuration is valuable for comparison purposes as well as for verifying the analytic model. This configuration has also several advantages over the LOCAL version, since it separates the client from the server, thus it gives a better overview of the actual behaviour of the Web service once deployed on the Web. It is particularly advantageous when using devices and nodes with limited memory, CPU resources and bandwidth.

3. **WAN (Web)**, where the client and server are located on the Web or on different domains. This configuration offers $70\ Kbit/s$ for download and $20\ Kbit/s$ for upload, which represent many legacy systems and data networks including wireless, mobile, sensor and residential networks.

The analysis is divided into two phases: online and offline. During the online phase, the Web service and the Web client application interact with each other in order to retrieve images, measure and save performance data in real time. During the offline phase, the data obtained during the online phase are utilised to calculate different performance parameters that can be plotted and submitted for further analysis.

4.6.1 Online phase of analysis

Once the Web service is being deployed and running, the Web client application offers two modes of operation: manual and automatic. In the manual mode, the client has to specify the names of the images to be retrieved one after another. On the other hand, the automatic mode retrieves all of the twenty images and processes them successively

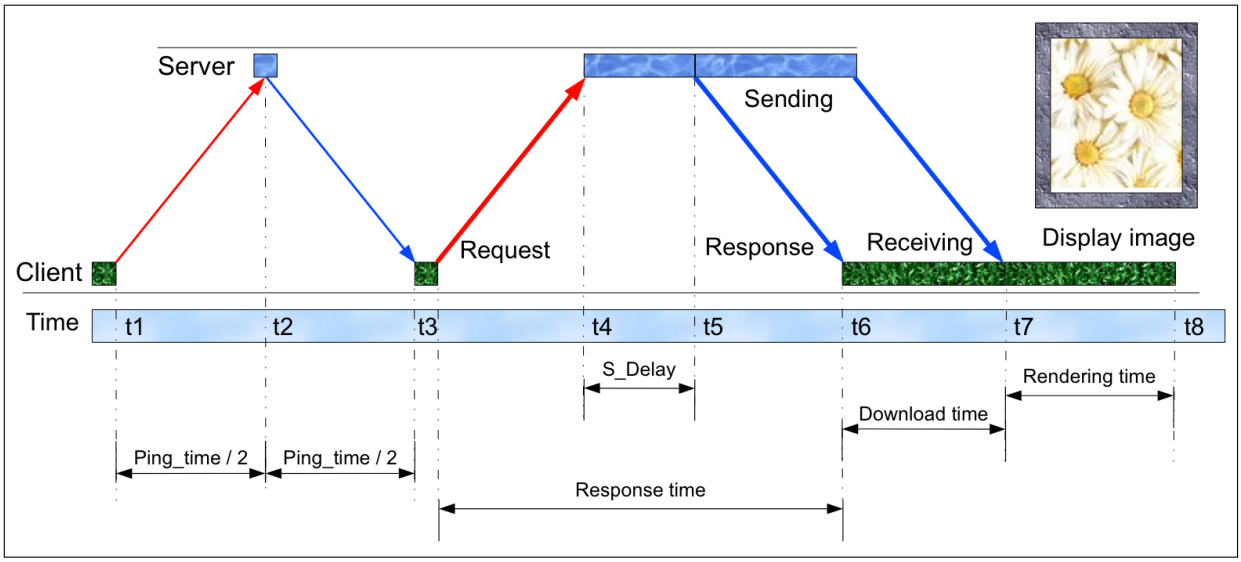


Figure 4.3 Explanation of relevant performance parameters for the image-retrieval Web service.

in a cycle of twenty iterations through the GUI. The number of desired cycles n is to be set before starting operation in the automatic mode. The Web service can provide a number m of images, up to twenty. To ensure the fidelity of data, we chose n to be five so that every performance parameter for each image is measured five times at five dispersed time intervals. The following pseudo-code 1 resumes the operations done by the Web client application:

Algorithm 1 Web client function(n, m)

Require: $\{n, m\} \in \mathbb{R}_{\geq 0}$, where $n = 5$, and m is the number of images

```
for j:=1 to n ; j++ do
    for k:=1 to m ; k++ do
        Query the k^th image
        if image is djvu-compressed then
            Decompress and save image
        end if
        Display image on the GUI
        Calculate performance parameters
        Display results
        Wait a few seconds
    end for
end for
```

As can be depicted from the above algorithm, the time needed to complete all iterations would be considerable for the WAN configuration, in particular, due to the low bandwidth.

The LOCAL configuration becomes handy during deployment and validation of the calculated QoS parameters, since it provides results quickly compared to the LAN and WAN configurations.

The Web client measures, records and calculates several relevant parameters as shown on figure 4.3, which constitute the input to the offline phase of analysis. It is worth noting that several other parameters are derived from those mentioned below, thus they are not mentioned for the sake of brevity:

- The ***ping_time*** (round-trip time), which is the time needed for a tiny packet (sent by the client) to reach the server and to return back to the client. This time is denoted as $t3 - t1$ on figure 4.3.
- The ***processing delay time (S_Delay)*** at the server $(t5 - t4)$, which is the time duration between receiving a request and sending a response.
- The ***response time*** $(t6 - t3)$, which is the time duration between sending a request at the client and receiving a response.
- The ***download time*** $(t7 - t6)$, which is the time between receiving (at the client side) the first and last bits of a response message.
- The ***image rendering time*** $(t8 - t7)$, which is the time between receiving the last byte of an image and displaying the whole image on the monitor (GUI).
- The ***image compression ratio***, which is expressed using the relative per cent error calculation. It describes the relation between an image's non-compressed (original) size os and its compressed size cs as follows: $\%image\ compression\ ratio = 100\%(os - cs)/os$. A positive image compression ratio, thus a desired value, means that the compressed size is less than the non-compressed size.
- The ***image-retrieval speed***, which is the image size divided by the download time.
- The ***actual download speed***, which is calculated by dividing the received data size (or response message size) by the download time.

4.6.2 Offline phase of analysis

As shown on figure 4.5, the percentage compression ratio (expressed as relative per cent difference) and the image size are not correlated. It is also noticed from the same figure that the compression ratio is higher than 40% for all of the images except for the

one with the largest size. Each image has different content, thus the compression ratio varies accordingly [66]. Images with high variations in colour and details have the least compression ratios of 8.6% as shown on figure 4.4b. On the other hand, images with large blocks of uniform colour have more redundant pixels and thus high compression ratio is attained. The highest compression ratio is 87.2% for the image with 655 bytes, since it has only one pixel and one colour. The second highest compression ratio is 81.0% as shown on figure 4.4a.

(a) Image with the second highest compression ratio 81.0%.

(b) Image with the lowest compression ratio 8.6%.

Figure 4.4 Images with high and low compression ratios.

Dealing with the effective compression ratio instead of individual ratios simplifies the analysis without loss of precision. Thus we deal with the image samples as a single large image.

The overall average image compression ratio can be obtained by dividing the area of the histogram on figure 4.5 by the range of image sizes. Let n and i be the number of images (i.e. $n = 10$) and the index of the current image, respectively, where $\{n, i\} \subset \mathbb{R}_{\geq 0}$. For each image$_i$, where $i \in [1, 10]$, we denote its size by Δx_i and the corresponding compression ratio by y_i. The *total area of all columns* $Area_{total} = \sum_{i=1}^{n}(\Delta x_i \cdot y_i)$ and the *total width of all columns* $Width_{total} = \sum_{i=1}^{n} \Delta x_i$. The global compression ratio (i.e. for all samples) is given by:

$$\%Compression = \frac{Area_{total}}{Width_{total}} \times 100\% \tag{4.1}$$

By substituting the expressions of $Area_{total}$ and $Width_{total}$ in equation 4.1 we obtain the global average compression ratio for our sample images as follows:

$$\%Compression = \frac{100\%}{\sum_{i=1}^{n} \Delta x_i} \sum_{i=1}^{n} (\Delta x_i \cdot y_i) \tag{4.2}$$

As shown on figure 4.5, the number of images is 10 and thus the global average image compression ratio is 30.21%.

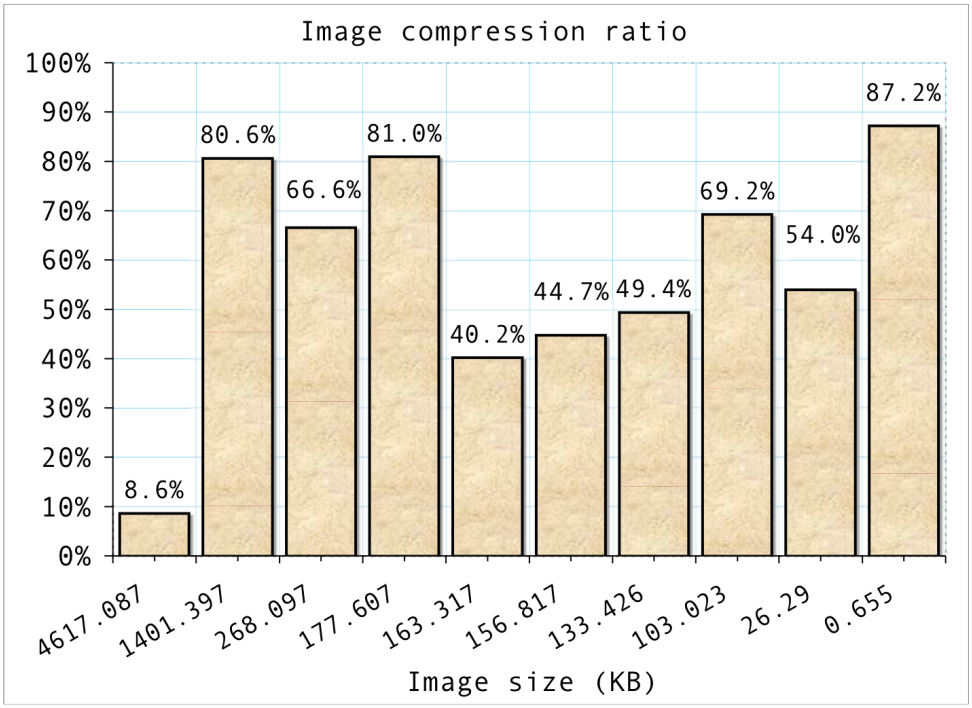


Figure 4.5 Individual image compression ratios.

The compression ratio influences the following parameters: ***response time***, ***download time*** and ***rendering time***.

Our practical results show that the *ping_time* does not change much for a given scenario and thus it can be considered constant. Moreover, with the assumption that multimedia Web services deal with large data sizes, the *S_Delay* (as shown on figures 4.6 and 4.7) is the dominant factor in the response time calculations. It is worth noting that the LOCAL configurations are for theoretical comparisons or proof of concept that indicates the coherence of the obtained results for the LAN and WAN configurations.

The overall download time of the whole image samples, i.e. 20 images, for the WAN configuration (as shown on figure 4.6) is the highest ($392.46\ s$) since it has the least

bandwidth of all scenarios. However, the S_Delay for the LOCAL configuration has the smallest value ($0.20\ s$), which is natural since the packets stay at the local host machine. The S_Delay for the LAN configuration is higher than that of the WAN configuration. This is justified by the higher CPU and memory resources available for the WAN configuration, for example both server and client run with more than $1.25\ GB$ of memory.

The percentage decrease in S_Delay and in response time on figure 4.8 due to compression for the LAN configuration is the highest of all configurations, 49.57% and 48.83%, respectively. This can be justified by the higher CPU resources present for the LOCAL configuration. The S_Delay and response time depend on several factors such as the CPU speed, operating system, memory and current running processes. The percentage decrease in response time is a little bit lower than that of the S_Delay, which is attributed to the ping_time (constant) that is cancelled out in the numerator during the calculation of the difference value, while the denominator becomes larger.

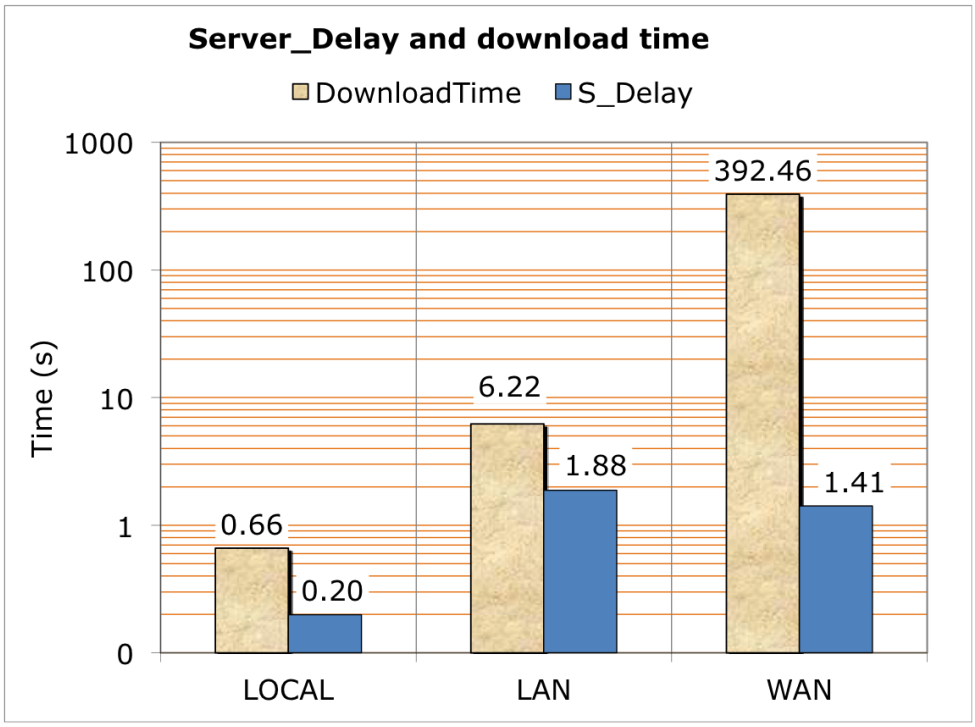


Figure 4.6 Server delay and download time for the whole image samples.

For WAN configuration, the *S_Delay* time is negligible, which is only 0.36% of the download time as can be noted from figure 4.7. On the other hand, the *S_Delay* is non negligible for both LAN and LOCAL configurations, 30.18% and 30.10%, respectively. This can be explained by the fact that the bandwidth available locally (for LAN and LOCAL configurations) is larger than that of WAN, which means that data transfer is higher and

thus download time is lower. It is noticed that the S_Delay for the WAN configuration is less than that for the LOCAL configuration since it depends on the hardware and CPU resources available within each configuration as show on table 4.1.

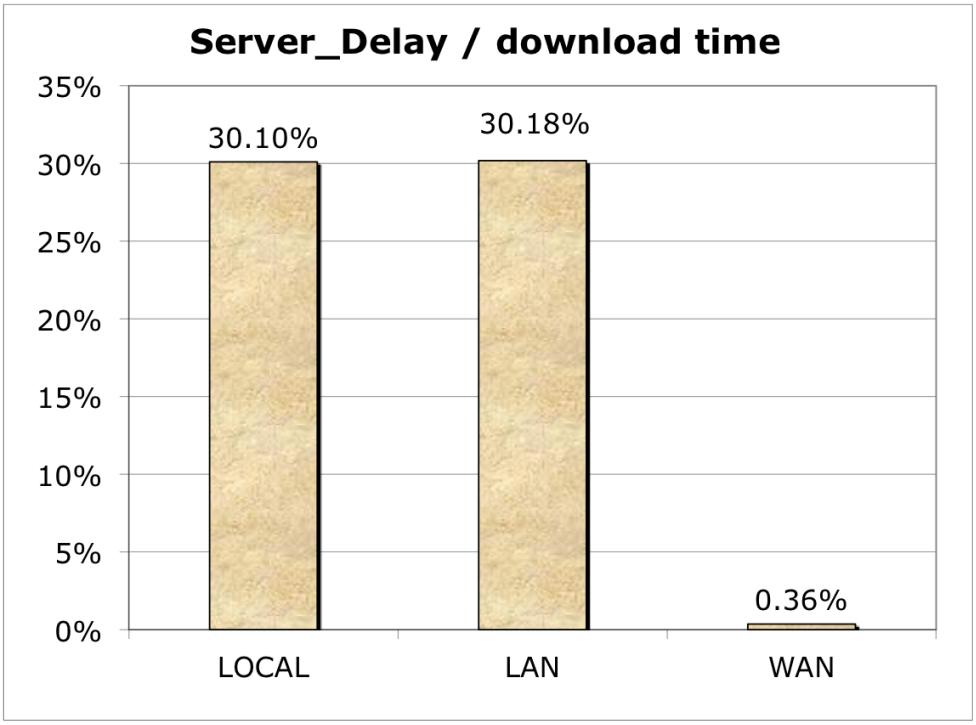


Figure 4.7 Server delay divided by download time.

As expected, the *S_Delay* decreases due to compression for the three scenarios (LOCAL, LAN and WAN), as shown on figure 4.8. The decreases in *S_Delay* (calculated using the per cent error calculation) for the LAN and LOCAL scenarios are close to each other (44.71% and 49.57%), while the decrease in *S_Delay* for the WAN scenario is 25.80%. The decrease in *S_Delay* for the LOCAL configuration is greater than that for the LAN scenario, since the resources are shared between the Web service and the Web client at the local host machine and thus the performance is affected.

The analysis results shown on figure 4.9 indicate that the download time is decreased by 30.52% for the WAN (Web) configuration. This decrease in download time is very close (within 1.03%) to the image compression ratio calculated previously (30.21%) in equation 4.2. Image compression reduces image size, thus fewer bytes are sent and less time is needed to download them at the client.

However, the download time decrease for LAN and LOCAL was not close enough to the theoretical value (30.21%), since some collected values were very small (sub-millisecond)

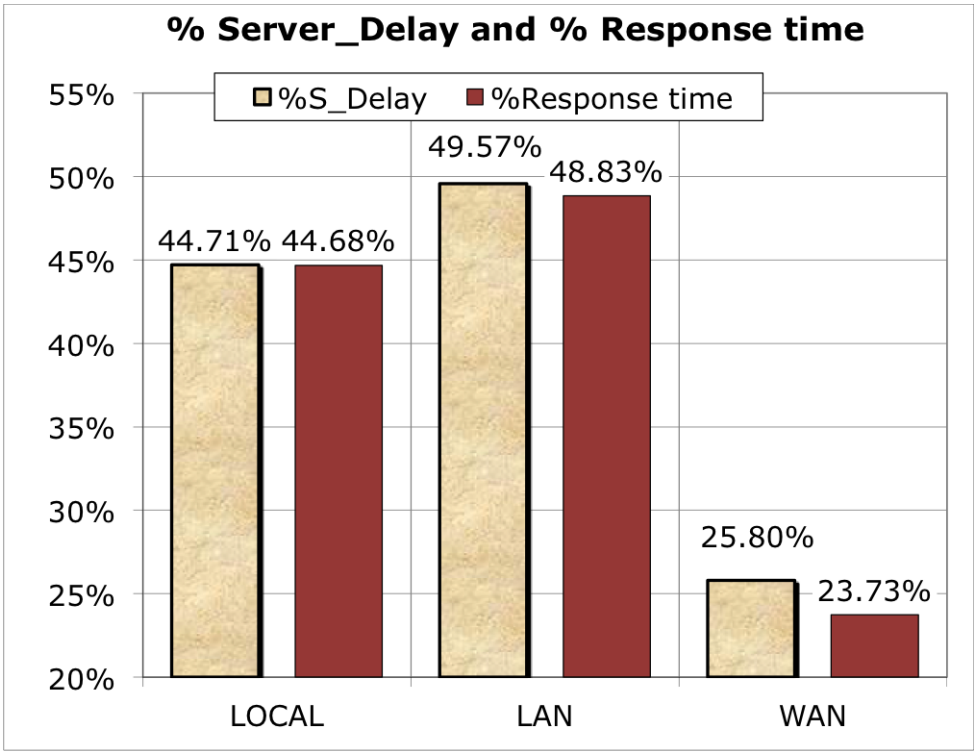


Figure 4.8 Decrease of server delay and response times due to compression.

and could be rounded off easily. The download time is much reduced in the LOCAL scenario where the Web client application and Web services codes are situated side by side and benefit from high-speed internal communication channels.

Figures 4.10, 4.11 and 4.12 show the various response times analysed for several configurations ($client \rightarrow server$) for LOCAL, LAN and WAN scenarios. For example, $Win \rightarrow Mac\ (WAN)$ configuration means that the client is running on Windows XP and the server is running on Mac OS X and both of them are situated in two different networks and two domains on the Web (as explained in section 4.5.1). In figures 4.10, 4.11 and 4.12, the response time is not linearly correlated to the image sizes, and thus the behaviour of multimedia Web services that deal with large payloads, for instance images, cannot be simply predicted by utilising analysis models that are based upon small-sized packets. Figures 4.10, 4.11 and 4.12 show that each curve tends to have three distinct regions: below $60\ KB$, within $60 - 300\ KB$ and above $300\ KB$, where the slope changes accordingly.

The non-linear behaviour of the response time is attributed to several factors, which can be summarised as follows:

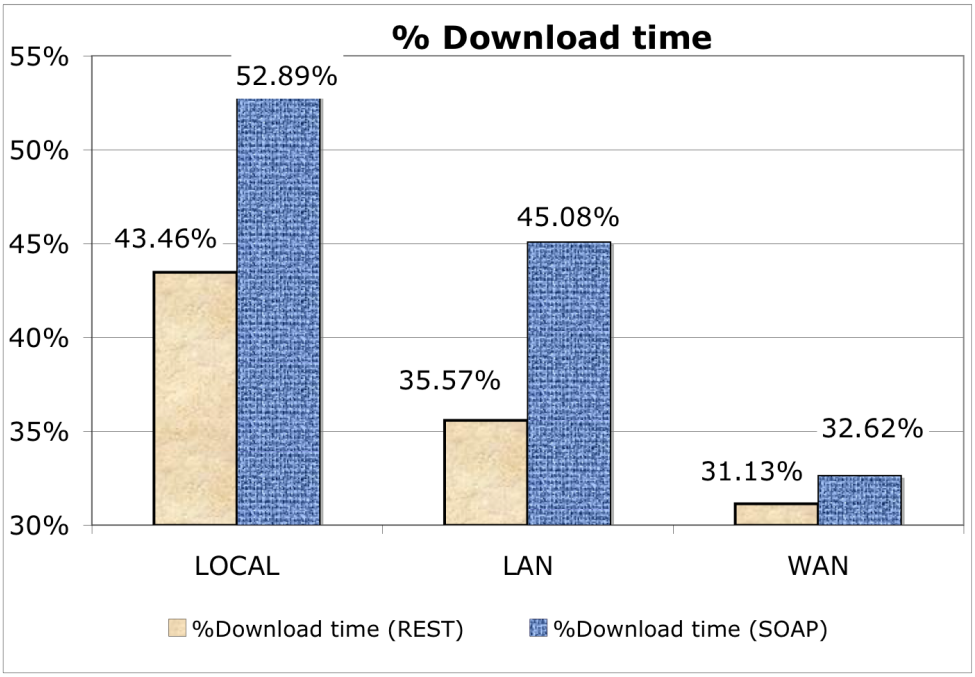


Figure 4.9 Decrease of download time due to image compression.

- The variation in network propagation delay and traffic congestion.
- The variation in CPU load. This factor is kept minimum as we utilised the Web server and Web client application in single user mode with minimum processes running. Running the complete set of iterations in a sequential manner and at the same period of the day can help keep the variations in network configurations to a minimum.
- The need for virtual memory for large payload sizes. The processing of large images takes considerable amount of RAM (*Random Access Memory*). The client application needs additional memory for the processing of received images and for the corresponding calculations. This in turn increases the response time.

Figure 4.10 shows that the response time is higher for the $Mac \rightarrow Mac\ (LOCAL)$ configuration, compared to the $Win \rightarrow Win\ (LOCAL)$ configuration, since the Windows machines have higher CPU and memory resources available, as shown in table 4.1.

Figure 4.11 shows that the $Linux \rightarrow Win\ (LAN)$ curve is below that of $Win \rightarrow Win\ (LAN)$ and $Win \rightarrow SUN\ (LAN)$, i.e. response time is lower for the $Linux \rightarrow Win\ (LAN)$ since the Linux machines have more effective processing resources and memory available (see table 4.1). It is noticed that best linearity or quasi-linearity is

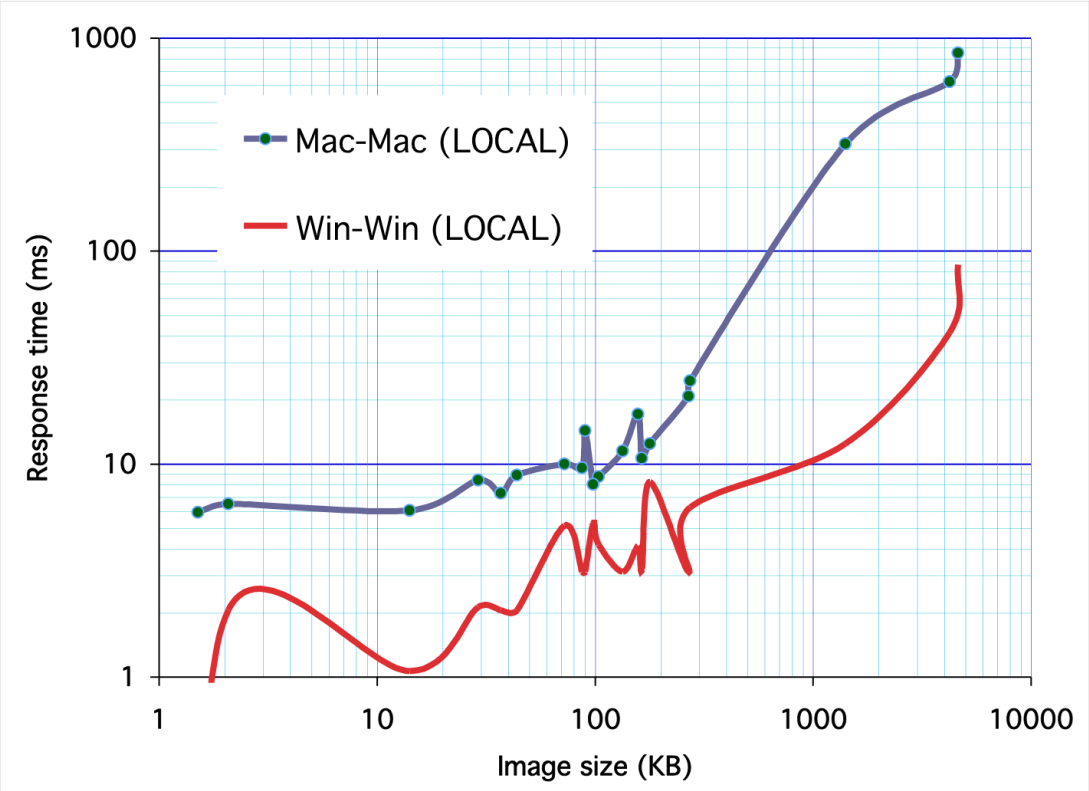


Figure 4.10 Response time for LOCAL configurations.

noticed for Linux and OS X ($Linux \rightarrow Win$ and $Mac \rightarrow Win$), while for Windows it was non consistent, which is related to the particularities of the Windows operating system.

Figure 4.12 shows that the response time values for the $Win \rightarrow Mac$ (WAN) configuration are about $20\ ms$ and are almost constant in the range of image sizes below $60\ KB$, which is explained by the fact that the *ping_time* becomes dominant in the calculation of the response time. The other factor in the calculation of the response time is the *S_Delay*, which is negligible for the image sizes below $60\ KB$.

We utilised different tools in order to analyse the internal composition of packets, which include the built-in HTTP monitor of GlassFish (accessible from within NetBeans), the service tester Web interface in NetBeans and Wireshark[6].

The last three rows on table 4.2 indicate that the header information for response messages is well above 10%, which correspond to image sizes of $12.096\ KB$ $(240 \times 180\ pixels$, compressed image), $655\ bytes$ $(1 \times 1\ pixel$, non-compressed image) and $84\ bytes$ $(1 \times 1\ pixel$, compressed image). This can be explained as follows: SOAP

[6]Wireshark is a general packet sniffing tool that is available from http://www.wireshark.org.

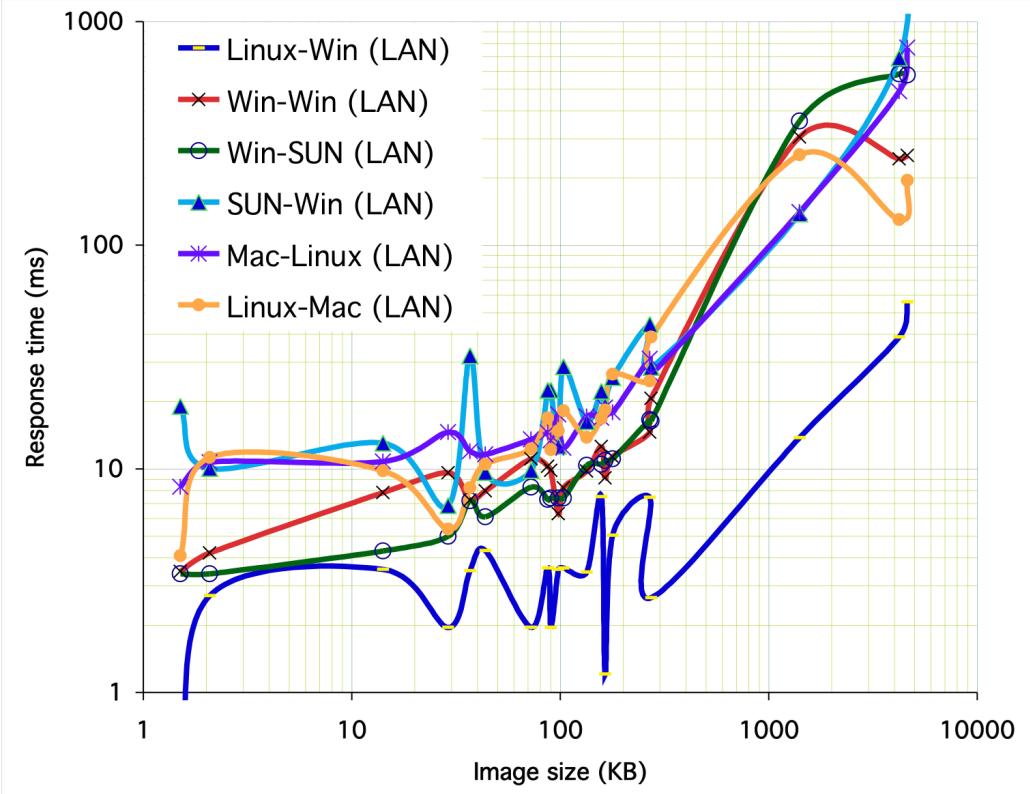


Figure 4.11 Response time for LAN configurations.

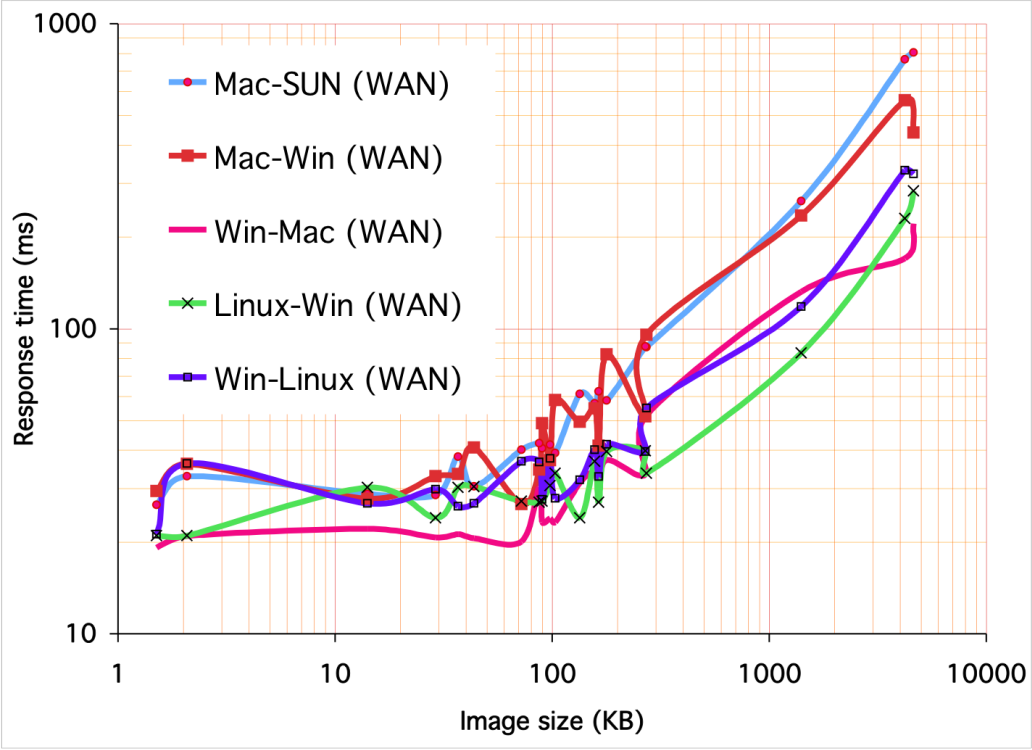


Figure 4.12 Response time for WAN configurations.

wraps the actual data into an XML envelope (SOAP overhead), augments the header information and thus the total size of sent information is increased. When the message size is relatively small, the ratio (header size)/(total message size) becomes relatively high. We consider that the SOAP overhead is small when the ratio is below 10%. When the envelope information becomes considerable, i.e. above 10%, compared to the response message size, the efficiency of SOAP becomes low.

Table 4.2 Response and request message packets details.

Response size (KB)	Payload (image) (KB)	Header size (KB)	Request (KB)	Header percentage (KB)	Efficiency (Y/N)
4617.359	4617.087	0.272	0.700	0.01%	Y
4218.623	4218.351	0.272	0.699	0.01%	Y
1401.673	1401.397	0.276	0.697	0.02%	Y
272.187	271.912	0.275	0.696	0.10%	Y
268.370	268.097	0.273	0.697	0.10%	Y
177.882	177.607	0.275	0.697	0.15%	Y
163.589	163.317	0.272	0.701	0.17%	Y
157.092	156.817	0.275	0.695	0.17%	Y
133.701	133.426	0.275	0.699	0.21%	Y
103.295	103.023	0.272	0.694	0.26%	Y
97.899	97.624	0.275	0.704	0.28%	Y
89.873	89.601	0.272	0.696	0.30%	Y
86.987	86.715	0.272	0.698	0.31%	Y
72.259	67.556	4.703	0.702	6.51%	Y
43.614	41.064	2.550	0.697	5.85%	Y
36.868	33.807	3.061	0.696	8.30%	Y
28.993	26.290	2.703	0.697	9.32%	Y
14.084	12.096	1.988	0.696	14.12%	N
2.074	0.655	1.419	0.699	68.42%	N
1.503	0.084	1.419	0.698	94.41%	N

The data in table 4.2 and the Web service tester interface is utilised to analyse the request messages in details. The request size is almost constant, approximately $700\ bytes$, for all images, since the only difference between requests is the image name. The Web service takes two arguments : the image name, for example, *tinyImage.jpg*) and the name of the method (*getImage*) to process the request. The total request size is $698\ bytes$ of which the SOAP part is $273\ bytes$ (encoded as XML with UTF-8) and the length of both arguments is 22 characters (bytes). The effective information in the SOAP message is only 22 out of 273 bytes, which constitutes $22/273 = 8.06\%$ of its size. Alternatively, to send 22 bytes, the SOAP overhead was $273 - 22 = 251\ bytes$, which constitutes $251/698 = 36\%$ of the request size. On the other hand, if we consider that the payload is the SOAP part ($273\ bytes$) then the efficiency of HTTP (transport protocol) and the underlying TCP/IP protocols becomes $273/698 = 39\%$.

4.7 Conclusion

Compressing payload in multimedia Web services greatly improves their QoS, enhances security and saves bandwidth. Local resources such as CPU and memory can be easily afforded and tuned in order to improve the QoS, while Internet resources such as bandwidth and remote processing are shared and vulnerable to congestion and delay, leading to QoS degradation. This emphasises the role of data and multimedia compression for Web services and other Internet applications.

We developed an analysis for the performance of multimedia Web services utilising binary data compression for three scenarios: LOCAL, LAN and WAN. The LOCAL configuration provides guidelines for the measured and calculated parameters as well as a rapid test and implementation on a single machine. In order to demonstrate our analysis method we implemented an image-retrieval Web service and a Web client application that measures and calculates various QoS performance parameters in real time. Our analysis is based upon realistic data model, where the payload is mainly multimedia data and the SOAP header information is negligible except for the smallest images in the sample. The analysis spans a wide range of image sizes from 1×1 pixel up to 3822×4819 pixels.

We have also demonstrated some further analysis including payload and data overhead, which are influenced by the choice of SOAP as a messaging protocol.

An important contribution of our approach is that it promotes the utilisation of data and multimedia compression techniques in Web services implementations. The analytic approach to Web service QoS quantification is based upon realistic data and representative data samples. We propose also including multimedia compression techniques such as DjVu in current Web standards such as HTTP. The proposed approach can be extended and adapted to accommodate the specific criteria imposed by LAN/WAN, wireless LAN, Ad-Hoc, mobile networks, IP-Telephony and distributed systems.

4.7.1 Future work

We plan to extend our approach to include more QoS parameters, more multimedia types such as video and voice. Also, we would like to extend our analysis to include RESTful Web services.

CHAPITRE 5

ANALYSE DE PERFORMANCE DE SERVICES WEB MULTIMÉDIA

Avant-propos

Auteurs et affiliation :

AL-CANAAN Amer ; étudiant au doctorat et chargé de cours au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

KHOUMSI Ahmed ; professeur au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

Date d'acceptation :

11 avril 2011.

État de l'acceptation :

version finale publiée.

Revue :

First International Conference on Wireless Communications and Mobile Computing (MIC-WCMC 2011).

Référence :

A. Al-Canaan and A. Khoumsi, «The Impact of Binary Data Compression on QOS and Performance of SOAP and RESTful Multimedia Web Services ». In Proceedings of Mosharaka International Conference on Wireless Communications and Mobile Computing (MIC-WCMC2011), Istanbul, Turkey : 3-5 June 2011, pages :77-83.

Titre français :

Effet de la compression binaire sur la qualité de service et sur la performance de services Web multimédia de types SOAP et RESTful

Contribution au document :

Cet article correspond aux contributions 2 et 3 qui se résument comme suit :

- Élaborer une liste de paramètres pertinents qui influencent la QdS selon les résultats obtenus lors de la mise en œuvre de SWM en utilisant le protocole SOAP et le style REST.
- Élaborer une analyse de performance de SWM basés sur le protocole SOAP et sur le style REST. L'analyse porte sur différents paramètres de QdS ainsi que différentes plate-formes et bandes passantes. Elle comporte l'utilisation de la compression binaire pour trois scénarios : local (à la machine même, appelé LOCAL), sur un réseau local (appelé LAN) et sur le réseau d'Internet (appelé WAN). Deux SWM ont été mis en œuvre avec une procédure spécifique permettant de cueillir les données de performance en temps réel.
  L'analyse permet ainsi d'étudier le comportement de SWM réels comportant des images de tailles diversifiées soient entre 1×1 pixel jusqu'à 3822×4819 pixels. L'analyse propose d'intégrer des méthodes de compression de données binaires telles que DjVu dans les standards du Web de manière similaire à HTTP.

  L'analyse est élargie pour couvrir d'autres paramètres de qualité de service liés à la vidéo et au son.

Résumé français :

Cet article présente une méthode d'analyse pour évaluer l'effet de la compression binaire de données sur la qualité de service et sur la performance de services Web multimédia de récupération d'images, de types SOAP et REST. La méthode d'analyse engendre quelques paramètres, tels que le temps de réponse, le délai du serveur, le délai d'affichage et le temps de télé-chargement. D'autre part, la méthode d'analyse présente une comparaison objective entre les performances et les QdS de services Web fondés sur SOAP et REST.

L'analyse est fondée sur des données réalistes recueillies depuis deux services Web mises en œuvre selon le protocole SOAP et selon le style REST. L'analyse a été faite pour deux domaines différents sur des plate-formes hétérogènes ayant des bandes passantes variées. Quelques paramètres influençant la QdS ont été sélectionnés et évalués comme le temps de réponse, le délai du serveur et le temps de télé-chargement.

Mots-clés : services Web SOAP, REST, performance, QdS, compression d'images, temps de réponse, délai du serveur.

Note :

À la suite des corrections demandées par les membres du jury, le contenu de cet article diffère de celui qui a été accepté.

5.1 Abstract

This paper presents an analysis method that evaluates the impact of binary data compression on the QoS and the performance of SOAP and RESTful image-retrieval multimedia Web services. The analysis method involves several parameters including response time, server delay, rendering time and download time. On the other hand, the analysis method provides an objective comparison between the performance and QoS of both SOAP and RESTful Web services.

In order to provide our analysis method with live and realistic data samples, we implemented an image-retrieval multimedia Web service for two different domains with

heterogeneous platforms and different bandwidths. Several QoS parameters are selected and calculated including response time, server delay and download time.

Index terms— SOAP Web services, REST, performance, QoS, image compression, response time, server delay.

5.2 Introduction

Multimedia Web services including SOAP (*Simple Object Access Protocol*) and REST (*REpresentational State Transfer*) are widespread and contribute to the congestion of traffic on the Internet. The availability and low cost of hand-held devices and wireless LAN has increased the demand on QoS and performance of multimedia Web services for personal and professional domains. Software and hardware solutions are developed in order to maintain reasonable performance and QoS while introducing new services. In this paper, we consider the impact of binary compression, which can be thought of as a software solution. On the other hand, hardware solutions include higher data processing power, higher bus speed, memory modules of low latency and large capacity. In general, software solutions are more flexible and less expensive.

As a preliminary step to improving the QoS and performance of multimedia Web services through binary compression, we undertake a thorough analysis and study of the impact of data compression on the performance and QoS of image-retrieval multimedia Web services.

The analysis method that was introduced by [16] is utilised to evaluate the impact of binary data compression on the QoS and the performance of SOAP and RESTful image-retrieval multimedia Web services. We have used the following parameters in our analysis: response time, server delay, rendering time, download time, image-retrieval time and compression ratio. Our analysis method provides a detailed comparison between the performance and QoS of both SOAP and RESTful Web services.

We implemented two image-retrieval multimedia Web services (SOAP and RESTful) in order to provide our analysis method with live and realistic data samples. The image-retrieval Web services are implemented on two different domains on two different LANs with heterogeneous platforms and different bandwidths. This is advantageous since it shows how each parameter varies with respect to the different characteristics, such as bandwidth, of the platforms and domains.

The analysis method shows precisely the contribution of image compression to improving the performance and QoS of multimedia Web services. It also presents a set of performance and QoS parameters that are relevant to multimedia Web services. It gives a general view of the performance and QoS of Web multimedia services and explains how these parameters are related to each other. Thus it is utilised to determine how the QoS and performance of Web services are to be improved. It is carried out on LAN and WAN networks in order to quantify the various performance and QoS parameters. Based upon the proposed analysis method, all relevant parameters (response time, server delay, rendering time, download time, image-retrieval time and compression ratio) are classified with respect to their influence on the performance and QoS.

We also explain that the analysis method can be extended in the following two ways. Firstly, it can be adapted for other types of multimedia services that incorporate voice and video. Secondly, it can be adapted for other types of networks, such as wireless LAN, Ad-Hoc, mobile networks, IP-telephony [15, 17] and distributed systems.

The main contribution of this paper is that the analysis method covers up both SOAP protocol and RESTful style Web services. The analysis results compare both SOAP and RESTful style Web services performance.

The sections of this paper are as follows. The related work is in section 5.3. Section 5.5 presents the image-retrieval multimedia Web services utilising both SOAP protocol and RESTful style. Section 5.6 presents the analysis method and explains how it can be extended. The results of the analysis method are discussed in section 5.7. Finally, the conclusion and future work are in sections 5.8 and 5.8.1, respectively.

5.3 Related work

A comparison is held between Web services and ASP (*Active Server Pages*) in terms of performance [111], showing how the data overhead of Web services messages is considerable, which decreases the throughput and increases the response time of Web servers. The article also compares the throughput of Web servers with and without payload compression. Unlike the work in [111], we utilise a wider range of binary data sizes, rather than a few Kbytes of XML data.

Web services' latency is compared with that of RMI in [39], showing that Web services outperform RMI when document-oriented style of communication is utilised, rather than

RPC style. As latency increases, the use of document-oriented communication style in Web services becomes more beneficial than RMI.

Not only does the performance of Web services depend on the implementation technology, such as RMI and SOAP, but also on the encoding of payload data [71] and on some technologies including tools for XML processing [68], which may make significant performance improvements. Generally, RMI is proved by several articles to be faster than SOAP. For a given Web service that is available with both RMI and SOAP, the authors propose an approach that determines when it is proper to utilise RMI and when it is more adequate to work with SOAP according to predetermined QoS parameters, such as availability, accessibility and performance.

As opposed to the works in [39, 68, 71], we consider Web services based upon SOAP and RESTful style and present a comparison between both types.

As QoS contract composition is part of Web services orchestration and choreography, the work in [96] proposes a probabilistic approach to QoS soft contract composition that is based upon probability distributions rather than fixed figures. The objective is to produce more natural and less pessimistic QoS contracts that can be generated using Monte-Carlo simulations. Unlike the approach in [96], our analysis provides direct evaluation to Web services QoS through the quantification of QoS as a function of relevant QoS parameters.

Reducing the message size through compression [30, 65] enhances the security and the QoS for mobile devices running XML-based Web services. The security is enhanced through XML data encryption. The authors argue that the utilisation of XML compression and then encryption reduces the transmission time as well as the CPU processing time. Compression reduces message size and thus leaves less data to encrypt. Binary XML compression produces a trade-off between response time and throughput. Compression reduces messages sizes and increases throughput, while response time increases due to compression and decompression overhead [30].

Compared to the works in [30, 65], our work concentrates on performance and compression of binary data as opposed to XML.

5.4 Data and image compression

Payload sizes of multimedia Web services can be considerably large. When dealing with high resolution images and high quality video, data sizes can be above $1\ megabyte\ (MB)$, which may increase packet congestion probability, low QoS and bandwidth overload. Two solutions come to rescue the situation, namely, increasing the bandwidth and utilising data compression.

The first solution may involve new hardware upgrade and new infrastructure, which may not be feasible without adding more investment cost. We believe that this solution is effective merely in the short run, since it would eventually lead to saturation. On the other hand, our tests amongst many others, reveal clearly a gap between the actual and the nominal bandwidth figures [72]. For example, for an Internet cable modem with nominal download rate of $512\ Kbit/s$, the actual data rate was only $75\ Kbit/s$.

The second solution is utilising data compression. Different data compression techniques can be utilised efficiently to help Web services keep up with the growing demand on multimedia Web contents and respect their QoS constraints. Several approaches propose compressing SOAP headers, which is less effective for multimedia Web services, since the length of SOAP headers is much smaller than the payload size.

Thus, the application of a high compression rate utility is desired so as to maintain the available Web bandwidth and QoS for multimedia Web services as well as for other types of services such as IP-telephony. The choice of a data compression technique is based upon the following criteria [85]:

- compression rate.
- speed of compression and decompression and,
- compatibility and ease of conversion between different formats.

Usually, binary compression libraries utilise one or more algorithms of compression to achieve higher compression rates. Some compression libraries make it possible to compress data that are already compressed such as the open source DejaVu library [45]. For example, JPEG images can be compressed and displayed efficiently as dvju format. However, other equivalent compression tools can be utilised for image compression.

Converting back to JPEG format is possible through the DejaVu tools, by first converting to TIFF format. Other tools and libraries can be utilised for this purpose, such as

ImageMagick, which is another open source project that is available for a variety of platforms including Unix, OS X and Windows [74].

For demonstration purposes, the main characteristics of two tools of DejaVu, namely *c44* and *bzz*, are compared to the characteristics of PPM [85] (*Prediction by Partial Matching*), with three orders, namely PPM(4), PPM(6) and PPM(16) as represented on Figure 5.1. The compression ratio ($bits/Byte$) for a given image is calculated as its compressed size in bits divided by its original size in bytes (the lower the better). The speed of compression and decompression are expressed as KB/s (the higher the better). The *c44* image compression tool is based upon the wavelet technology (suited for only image compression), while the *bzz* and the PPM encoding utilities are general purpose since they can be used to compress text and binary image data.

In our test results on Figure 5.1, the PPM(6) has the best (i.e. highest) compression and decompression speeds while *c44* has the best compression ratio. As the order of PPM gets higher up to 16, the compression and decompression speeds become lower due to the increasing number of iterations of the PPM algorithm. Figure 5.1 shows also that the decompression of the *bzz* tool is more sensitive to machine processing capabilities compared to compression (the test was held with two different machines: OS X having more resources, in terms of CPU and memory, and Linux with fewer resources).

5.5 SOAP and RESTful image-retrieval multimedia Web services

In order to help quantify the influence of data compression on multimedia Web services more efficiently, we developed two image-retrieval Web services based upon SOAP (with MTOM enabled[1]) and RESTful style, which provide input data needed for our analysis. The data are collected in real time while the Web services are running.

This section provides some details about the image-retrieval Web services and the Web clients that we have developed. The SOAP image-retrieval Web service in this section is an improved version of the developed image-retrieval multimedia Web service in [16], while the RESTful multimedia Web service is created independently from scratch. The improvement has been done in two ways: optimisation of Java code and optimisation of the algorithm (pseudo-code) to collect and calculate parameters more efficiently.

[1] MTOM: *Message Transmission Optimisation Mechanism.*

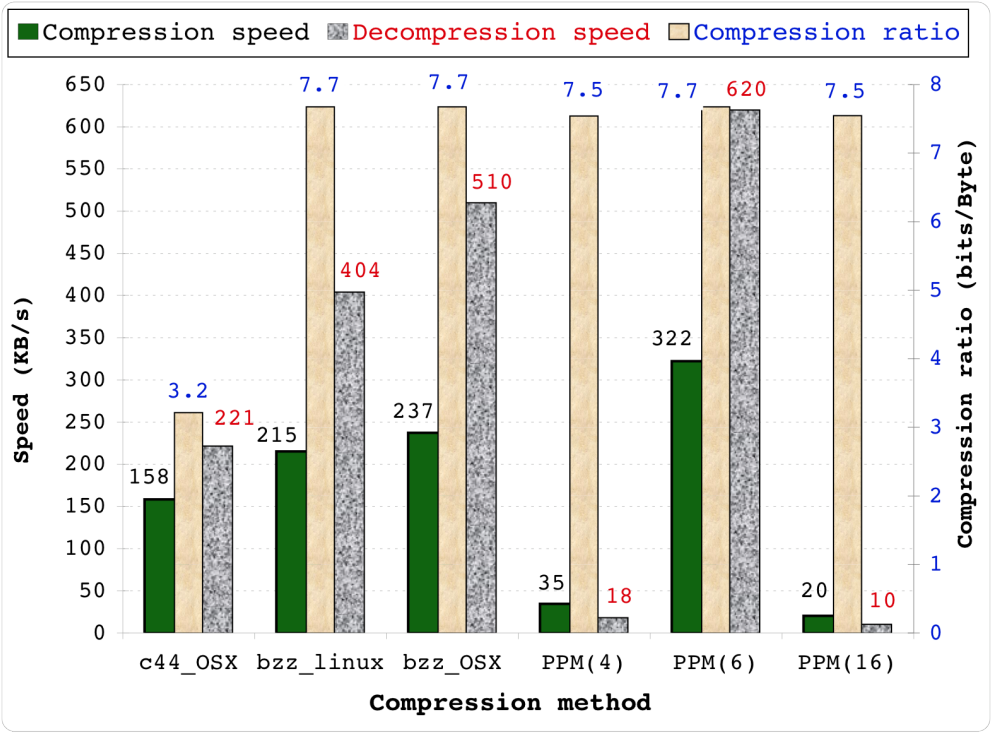


Figure 5.1 Comparison of some image compression and decompression tools.

Both Web services provide the clients with the same set of information and images in single-user mode.

The development has been preformed on heterogeneous platforms, where the role of Java comes into play: jdk1.5 on OS X (G5) and jdk1.6 on Linux, Solaris and Windows XP. The IDE (*Integrated Development Environment*) utilised to implement the Web service and the Web client application is NetBeans v6.71 with GlassFish server v2.1. Several tutorials on Web service implementation using NetBeans are available on the Internet.

Compared to the previous work in [16], we provide an improved analysis method in several ways. Firstly, we include both SOAP protocol and RESTful style, which allowed us to perform a detailed comparison between both types of Web services. Secondly, we have increased the amount of data in the analysis by adding a new machine running Windows XP, which rises the total number of machines to 6 instead of 5, distributed on two domains: domain 1 and domain 2. Each domain has 3 machines: domain 1 refers to a $10\ Mbps$ LAN, while domain 2 refers to a $1\ Mbps$ LAN. Thirdly, we have carried out a memory upgrade for one machine in domain 2 from $0.5\ GB$ to $2.5\ GB$ and an upgrade in network configuration for domain 1, where the download link has been upgraded: from

$0.512\ Mbps$ up to $1\ Mbps$. These upgrades reduce the time needed to obtain data results. Fourthly, we added a new performance parameter (image-rendering time at the client).

To analyse the responses between server and client application, we utilised different tools including the built-in HTTP monitor of GlassFish (accessible from NetBeans), the service tester Web interface in NetBeans IDE and Wireshark[2]. We also utilise Wireshark to verify that the MTOM is enabled as expected by checking the contents of the response messages.

At the client side, we developed two comprehensive client applications that perform two sets of functions:

1. retrieve and display a group of 20 images (as provided by the Web server) on a specially designed GUI, not shown here for the sake of brevity.
2. measure, record and calculate relevant QoS parameters.

The following QoS parameters are relevant to the binary data compression in multimedia Web services in this paper:

- the ping time, which is the time needed for a tiny packet of data to reach its destination and to return back.
- the response time, which is the time duration between sending a request packet of data and receiving its response.
- the processing delay time at the server, which is denoted as *S_Delay*. During this time, the server processes the request, fetches the image bytes and prepares them for delivery.
- the image retrieval time, which is the time to retrieve an image (ignoring the header information of data packets).
- the download speed, which is the actual speed of the link between the client application and the Web service.
- the compression ratio, which is a measure of the decrease in size for an image, represented using relative per cent difference calculations.
- the image rendering time, which is the time needed for the client application to display the received image bytes on the computer screen.

[2]Wireshark is a general packet sniffing tool that is available from www.wireshark.org.

The Web server provides four different types of responses to the client. Firstly, the requested binary bytes of an image in two possible formats: JPEG and DjVu [45] format, which is a high performance digital document format. Secondly, the time during which the server processed the image request. Thirdly, a list of all available names of stored images. Fourthly, information about the platform of the server machine.

Our Web service and Web client were developed for several platforms (see Table 5.1) that are located at two different domains:

- domain 1, with download speed of $1\ Mbps$: Mac OS X and Linux.
- domain 2, with download speed of $10\ Mbps$: Windows XP and Solaris.

Table 5.1 Systems configurations and platforms specifications.

Domain	Platform	RAM, CPU@clock frequency
1	OS X 10.5	2 GB, G5 @1.6 GHz
	Linux 2.6	2 GB, G5 @1.6 GHz
	Linux 2.6	386 MB, PIII @0.7 GHz
2	Windows XP	2.5 GB, PIV @2.5 GHz
	Windows XP	512 MB, PIV @2.5 GHz
	Solaris	512 MB, UltraSPARC @0.4 GHz

5.6 Performance analysis method

In this section, we analyse the performance of the multimedia Web services that we have developed and presented in the previous section 5.5. The two Web services provide a total of 20 images: 10 JPEG images and 10 compressed images with DjVu format. The image samples are the same as in [16]. The compressed images have been produced offline by the *c44* tool, which is provided by [45]. The selected images sample span dimensions from $1 \times 1\ pixel$ up to $3822 \times 4819\ pixels$, corresponding to 655 bytes up to $4617.087\ KB$.

The analysis is realised in the following configurations:

1. **LOCAL**, where the client and server code share the same resources of one machine. This configuration allows us to do preliminary verification before deploying the actual implementation on the Web, since the delay times and response times are lower.
2. **LAN**, where the client and server are situated in one LAN (one domain). Two separate LANs and thus two domains were available to our tests: domain 1 with three platforms (two Windows XP and one Solaris) and domain 2 with three platforms (one OSX 10.5 and two Linux 2.6, as shown on Table 5.1). This configuration is valuable for comparison purposes as well as for verifying the analytic model. This configuration also has several advantages over the LOCAL version. It offers a better overview of the actual behaviour of the Web service once deployed on the Web. This configuration also is particularly adequate when using devices and nodes with limited memory, CPU resources and bandwidth.
3. **WAN (Web)**, where the client and server are located on the Web. This configuration was tested between domains 1 and 2.

The Web client applications have two modes of operation: manual or automatic. In the manual mode, each image has to be specified and retrieved separately. In the automatic mode, the number of analysis iterations n is set in advance and all of the 20 images (compressed and non-compressed) are retrieved sequentially. To ensure the fidelity of data, we chose n to be 5 so that every performance parameter of each image is measured five times in dispersed time intervals. The pseudo-code in Table 5.2 describes the algorithm to retrieve the images automatically and to provide data for further analysis:

The Web client application calculates the following relevant parameters that constitute the input to the analysis method (see Figure 5.2):

- the ***ping_time*** $(t3 - t1)$, which is the time needed for a tiny packet (sent by the client) to reach the server and to return back to the client.
- the ***response time*** $(t6 - t3)$, which is the time duration between sending a request and receiving a response.
- the ***server delay time*** $(t5 - t4)$, denoted as *S_Delay*, which is the time duration between receiving a request (at the server side) and sending a response to the client.
- the ***image retrieval time*** $(t7 - t6)$, which is the time between receiving (at the client side) the first and last bits of an image (ignoring the header information).

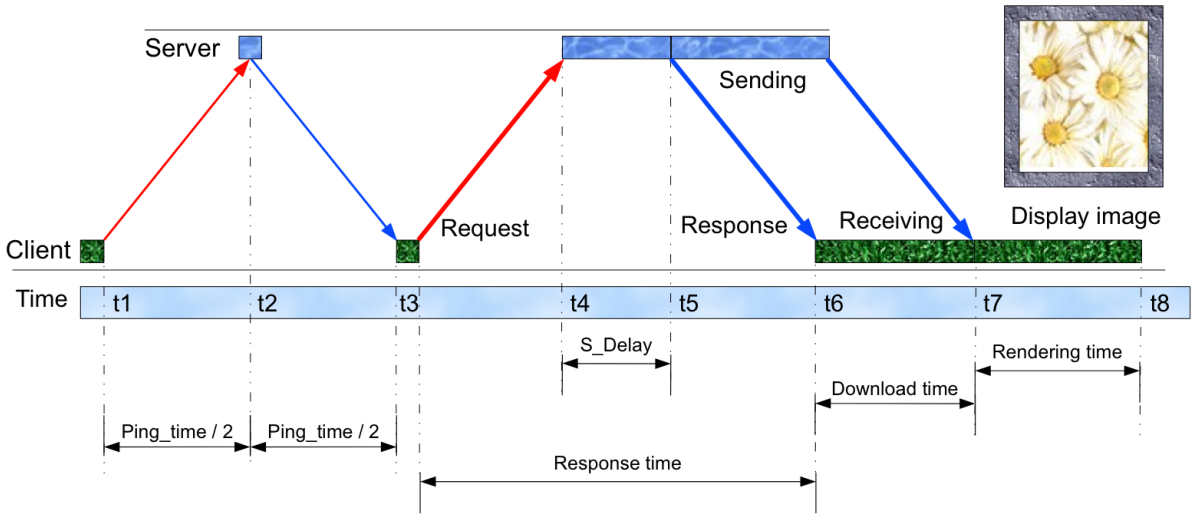


Figure 5.2 Representation of relevant performance parameters.

- the ***image rendering time*** ($t8 - t7$), which is the time between receiving the last byte of an image and displaying the whole image on the monitor.
- the ***image compression***, which is a measure of change in image size due to compression and is represented as a relative per cent difference value.
- the ***image retrieval speed***, which is the image size divided by the download time.
- the ***actual download speed***, which is the speed of receiving data (KB/s) and is calculated by dividing the received data size (including payload and header data) by the time during which data are downloaded.

In order to compare between the calculated QoS parameters of each image and those of its compressed version, the image rendering time, S_Delay and the download speed are expressed as relative per cent difference values.

5.7 Results and discussion

In this section we provide the analysis results of the impact of image compression on both SOAP and RESTful multimedia Web services in three configurations: LOCAL, LAN and WAN.

Table 5.2 Pseudo-code for the algorithm of the analysis method.

For $j = 1$ to n **do**
For $k = 1$ to 20 **do**
IF $j = k = 1$ (only once, at the beginning)
► Calculate the round trip time (ping).
► Obtain the server ID (information).
End IF
► Query the k^{th} image by name.
► Decompress if necessary and save image.
► Display the image.
► Calculate and display relevant data.
► Wait for a few seconds.
End For
End For

The average compression ratio for the whole sample is 30.21%, which is calculated by integrating the histogram and dividing the result by the sum of the total size of all images as in [16].

The *S_Delay* is decreased, which indicates that the performance is improved due to compression as shown on Figure 5.3 for both SOAP and RESTful multimedia Web services. The RESTful Web service has higher decrease values (better improvement) than SOAP for all configurations.

Since the download time is much lower for the LOCAL and LAN configurations, the ratio of $S\_Delay/download\ time$ becomes higher, which is above 38.1% as shown on figure 5.4. On the other hand, this ratio is less for the WAN configuration due to the higher download time compared to the two other configurations. The ratio is higher for REST in both configurations LAN and WAN, which indicates that REST offers best performance.

The analysis results shown on Figure 5.5 indicate that the decrease in download time is 31.13% and 32.62% for the WAN configuration. This decrease in download time is close (within 3% to 8%) to the image compression ratio (decrease in image sizes) calculated previously (30.21%), as expected. The image compression reduces image size, thus less

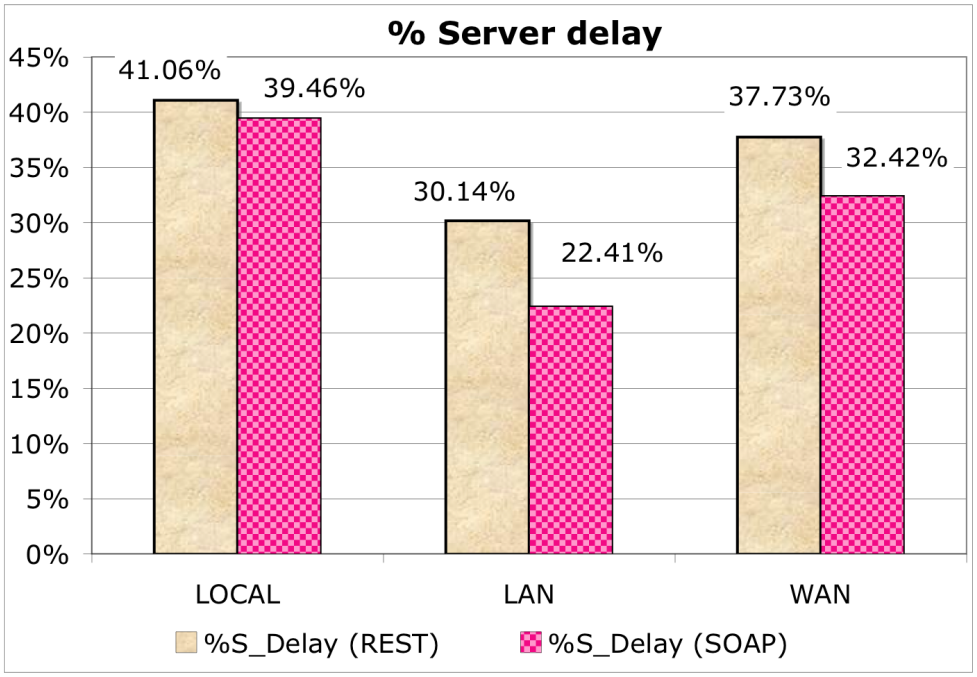


Figure 5.3 The decrease in server delay time due to compression for both SOAP and RESTful Web services.

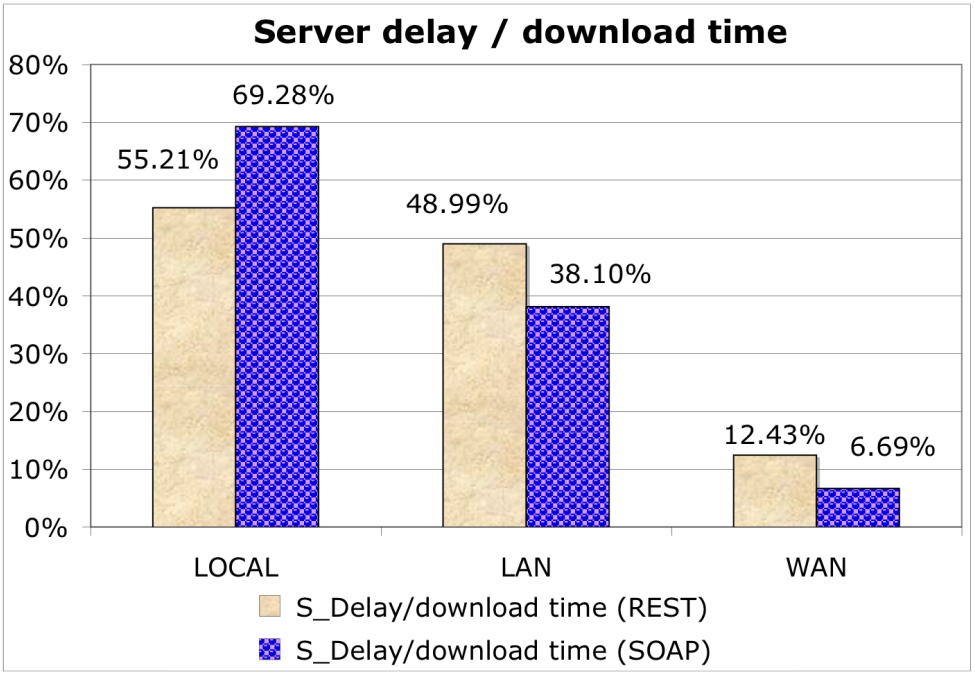


Figure 5.4 The ratio of *S_Delay/download time* for SOAP and RESTful Web services.

bytes are sent, which reduces the download time at the client. However, The decrease in download times for SOAP and RESTful Web services is close to each other for the WAN configuration (usual configuration for a Web service), while it is more significant for the SOAP Web service in LAN and LOCAL configurations.

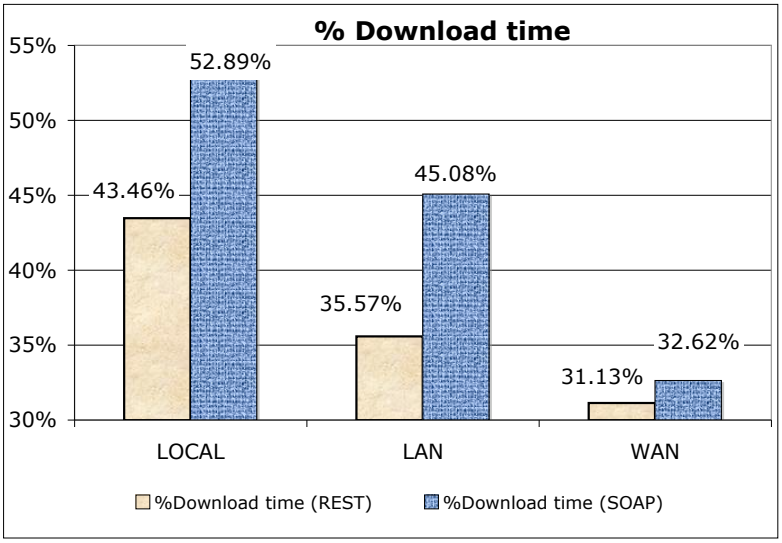


Figure 5.5 The decrease in download time due to image compression.

The rendering time is decreased significantly for all configurations as shown on figure 5.6. The decrease in rendering time is more significant for RESTful compared to that of SOAP Web service, which implies that the performance improvement is higher for the RESTful Web service implementation.

On Figures 5.7, 5.8, 5.9, 5.10 we show the various response times of the SOAP and RESTful Web services analysed only for the WAN configuration, for the sake of brevity. The notation $Win - Mac\ (WAN)$ means that the client application is run on Windows XP and the Web server is run on Mac OS X on two different networks (WAN configuration).

The response time is non-linear as shown on figures 5.9 and 5.10. Each curve tends to have three distinct regions: below $100KB$, $100 - 200KB$ and above $200KB$, where the curves rise up exponentially. This indicates that the response time increases as the size of the payload increases. The response time for the RESTful Web service is lower than its SOAP counterpart as shown on the figure. For the largest image sizes, REST offers response times below $200\ ms$ on eight curves compared to two for SOAP. Also, another important aspect of REST style is that the Web application client is implemented with less RAM: $256\ MB$ for SOAP versus $200\ MB$ for RESTful Web service, which implies that the overall performance gain with REST style is better compared to that with SOAP.

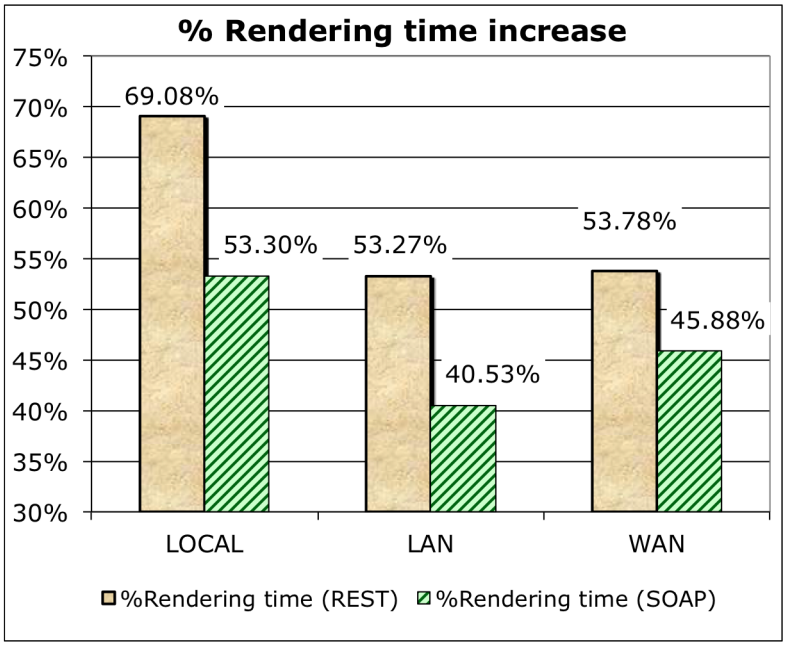


Figure 5.6 The effect of image compression on image rendering time.

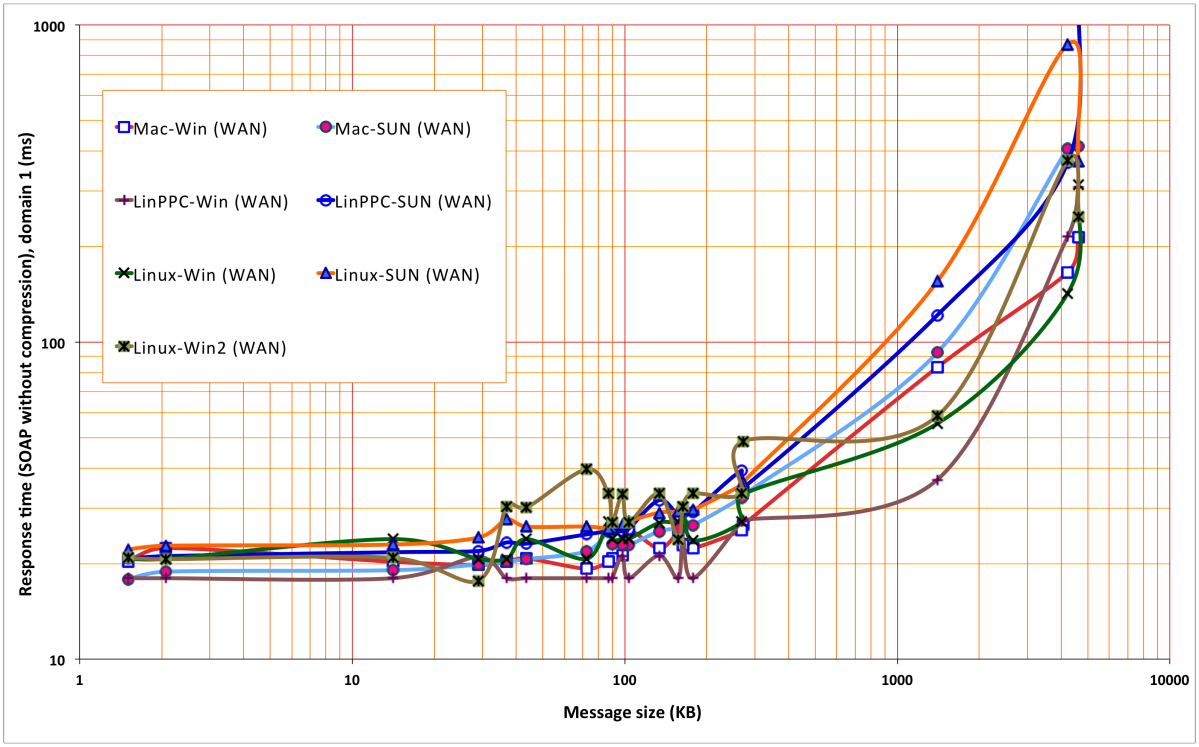
It is worth noting that the different measurements for SOAP and REST were taken on different occasions, which explains the variations between results in the same category.

A recapitulation of all parameters of QoS studied so far, in sections 4.6.1, 4.5.1 and 5.5, are presented in Table 5.3.

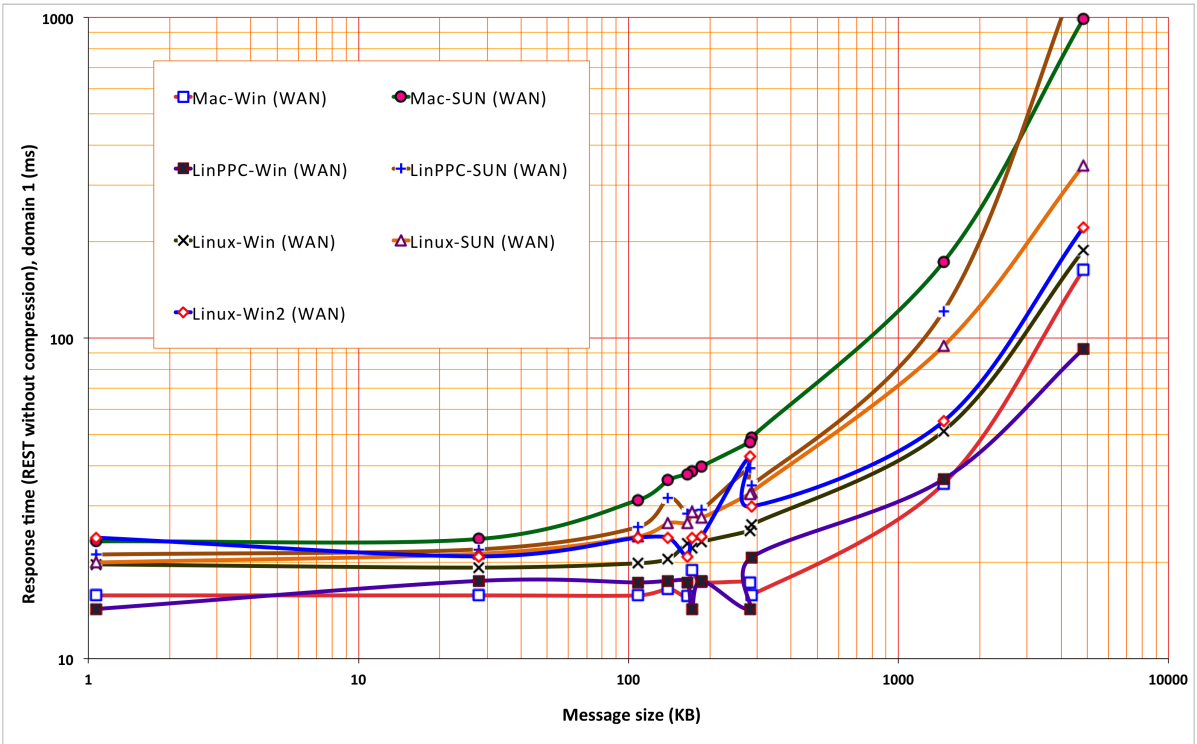
5.8 Conclusion

Compressing payload in multimedia Web services greatly improves its QoS, security and saves bandwidth. Local resources (CPU, memory) can be easily afforded and tuned, while Internet resources (such as bandwidth and processing) are shared and vulnerable to congestion and delay, leading to QoS degradation. This emphasises the role of data and multimedia compression.

In this paper we have developed two Web services utilising SOAP protocol and RESTful style in order to study the effect of binary data compression on multimedia Web services. Compared to our previous work in this field, we have added a new performance parameter (rendering time) to our analysis method. We have also improved the analysis algorithm and increased the amount of the input data. We have also introduced RESTful style in our study and have compared both in terms of performance. An important contribution

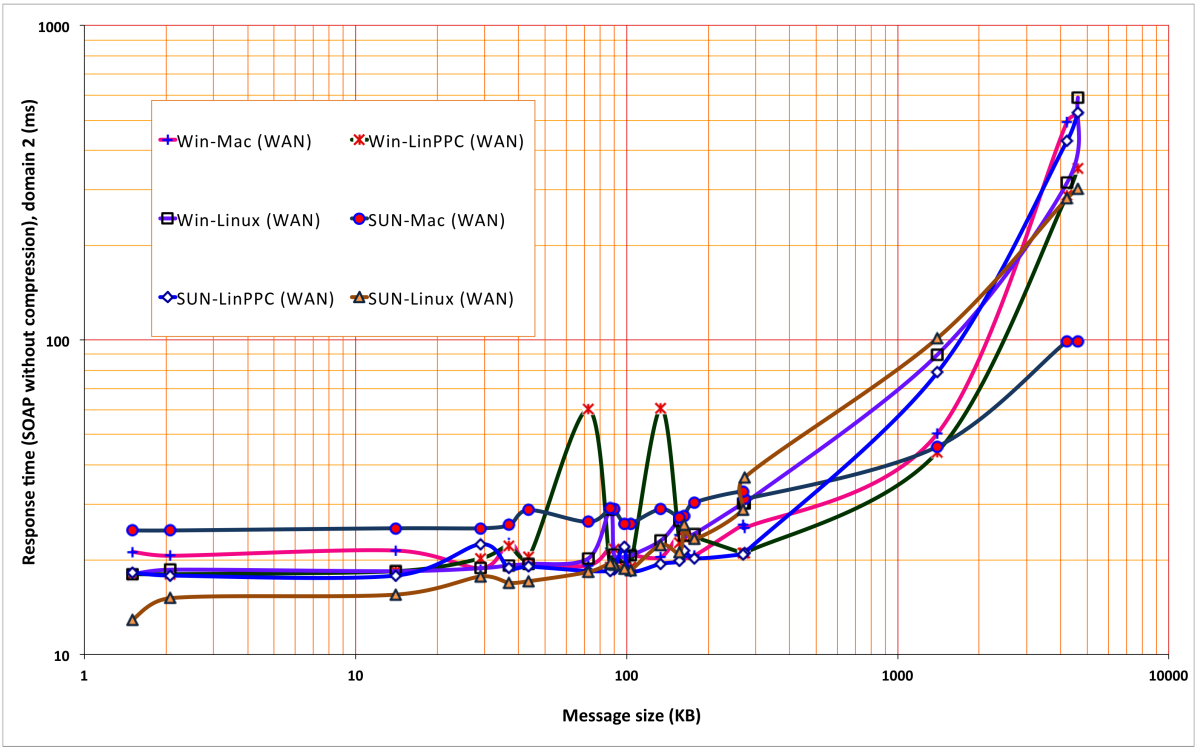


(a) Response time for SOAP without compression in domain 1.

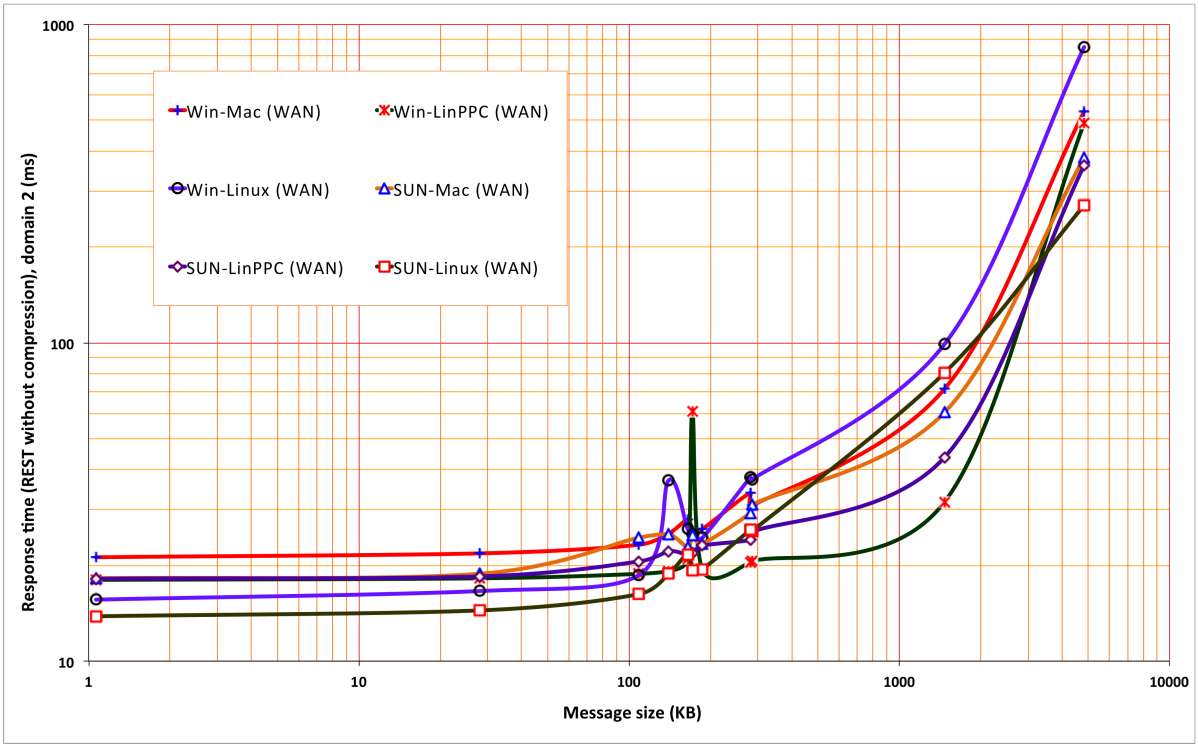


(b) Response time for REST without compression in domain 1.

Figure 5.7 Response time for SOAP and RESTful Web services without compression in domain 1.

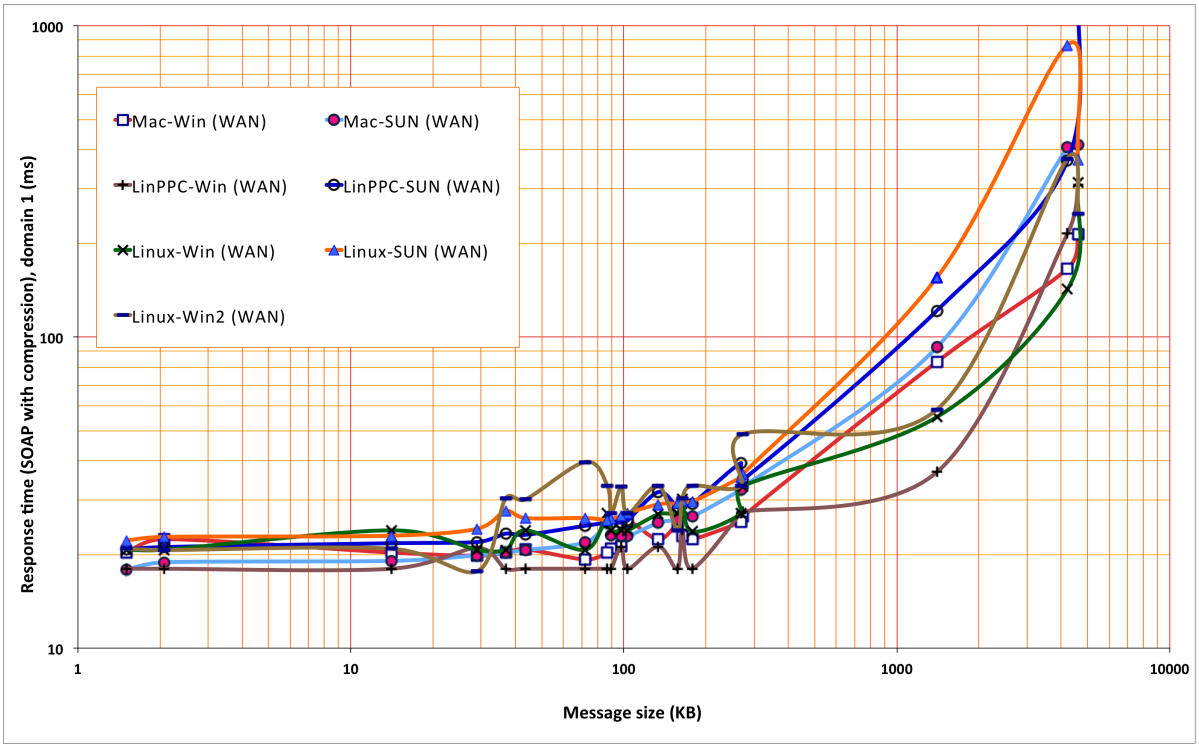


(a) Response time for SOAP without compression in domain 2.

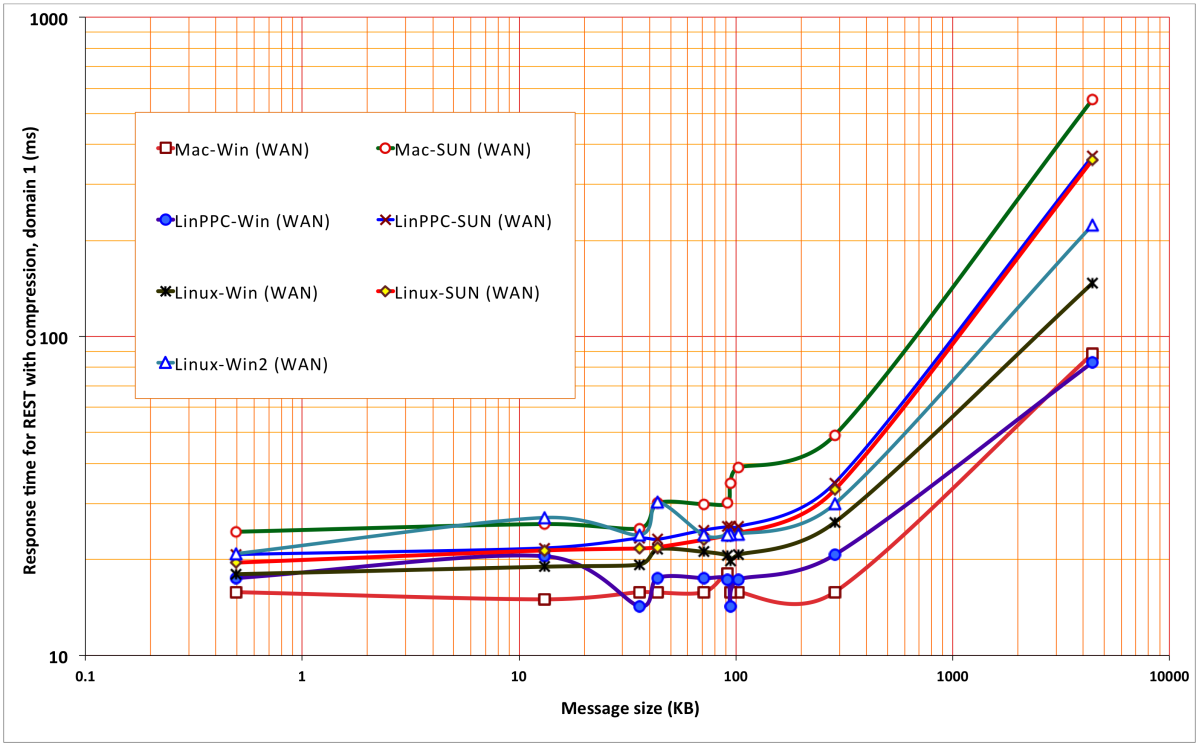


(b) Response time for REST without compression in domain 2.

Figure 5.8 Response time for SOAP and RESTful Web services without compression in domain 2.

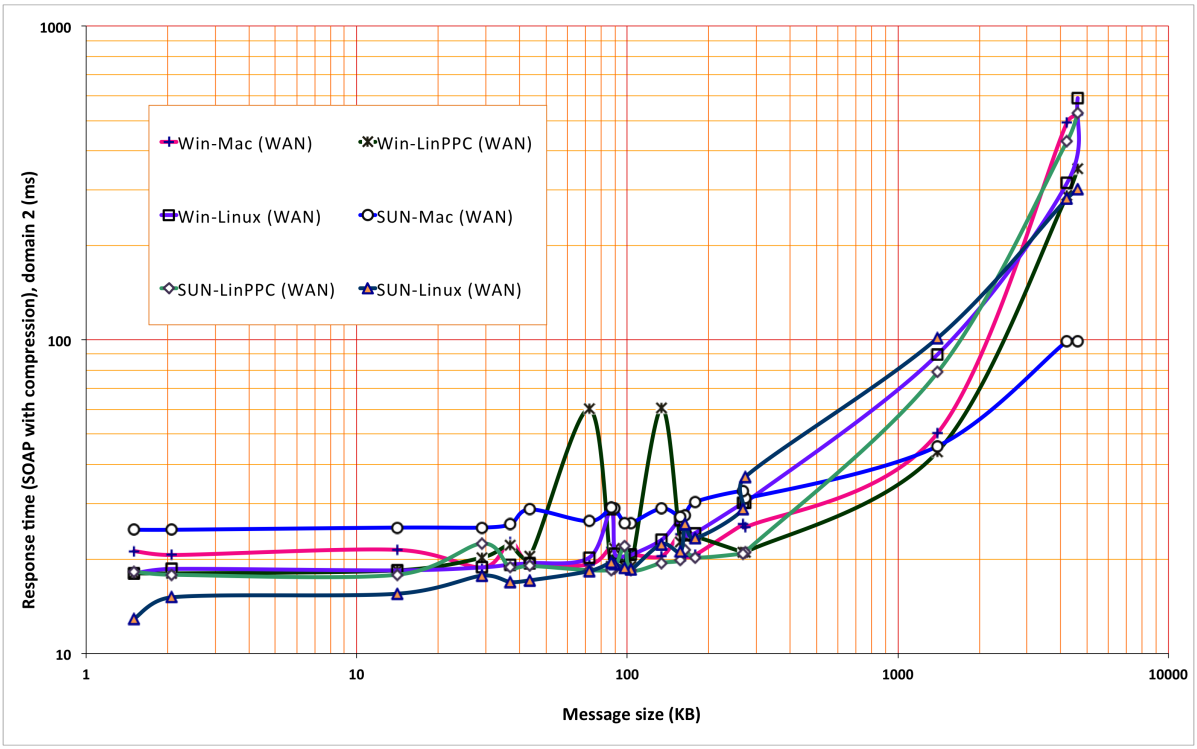


(a) Response time for SOAP with compression in domain 1.

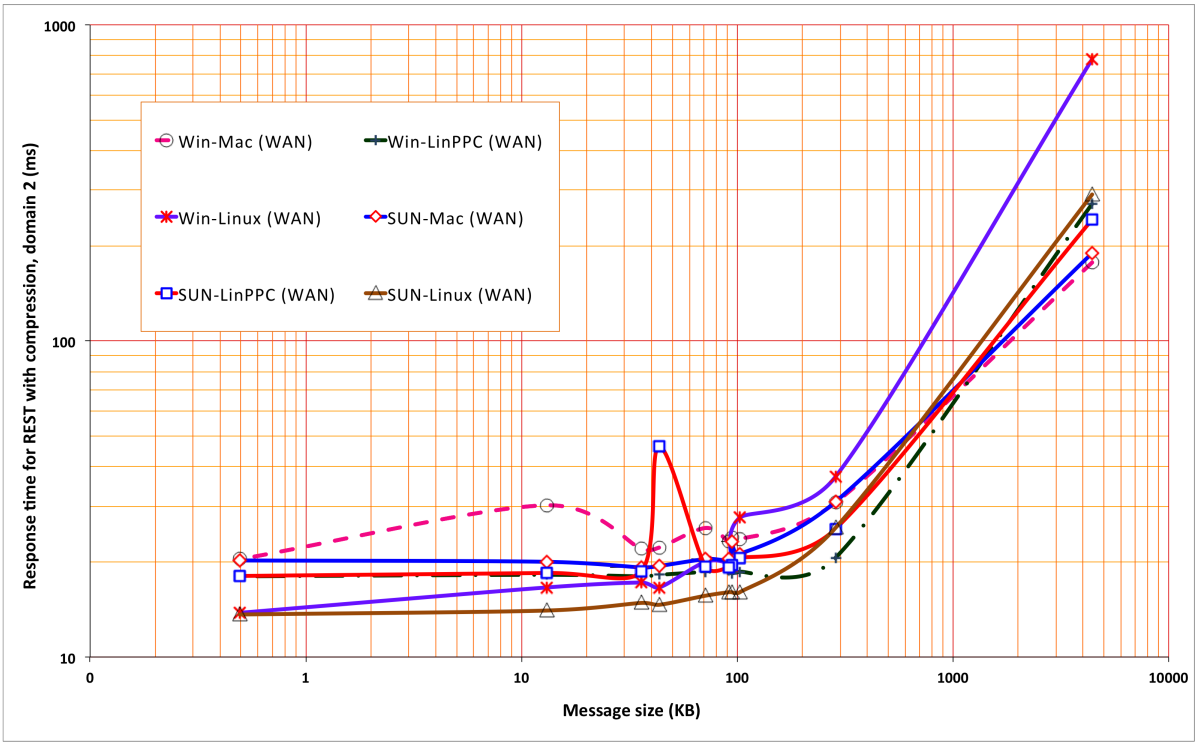


(b) Response time for REST with compression in domain 1.

Figure 5.9 Response time for SOAP and RESTful Web services with compression in domain 1.



(a) Response time for SOAP with compression in domain 2.



(b) Response time for REST with compression in domain 2.

Figure 5.10 Response time for SOAP and RESTful Web services with compression in domain 2.

Table 5.3 Recapitulation of relevant QoS parameters that influence MWS.

	QoS parameters[1]	**Summary**
1	Ping time	$t_3 - t_1$ on figure 5.2. Known as the RTT (Round Trip Time). Depends on the number of hops between client and server. Included in *response time*. Considered constant[2] if the client and server are not mobile.
2	Processing delay	$t_5 - t_4$ on figure 5.2 (of considerable effect on MWS). Included in the *response time*. Depends on processing power and memory on the server side.
3	Response time	$t_6 - t_3$ on figure 5.2 (of considerable effect on MWS). Can be improved by improving processing power[3], memory on the server side and binary data compression. Nonlinear behaviour for wide-range binary content, as shown on figures 5.7, 5.8, 5.9 and 5.10.
4	Download time	$t_7 - t_6$ on figure 5.2 (of considerable effect on MWS). Can be improved by applying binary data compression, reducing message (protocol) overhead, increasing bandwidth and increasing processing power. *Image retrieval rate* $\subset$ *Actual download rate* $\subset$ *Download time*
5	Actual download rate	
6	Image retrieval rate	
7	Image rendering time	$t_8 - t_7$ on figure 5.2. Can be improved by increasing data processing power and memory at the client side.
8	Image compression ratio	(of considerable effect on MWS). Can be improved by applying high quality and efficient compression[4] schemes.

[1] For almost all tested QoS parameters, the RESTful MWS outperformed their SOAP MWS counterparts.

[2] RTT is small and can be negligible compared to other QoS parameters in the context of MWS.

[3] Processing power is influenced by other factors, such as computer architecture and operating system configuration.

[4] Compression of data also improves traffic balancing and reduces congestion on the Internet.

of our approach is that it promotes the utilisation of data and multimedia compression techniques in Web services implementations (SOAP and RESTful) in order to improve the performance and QoS. We also propose an analytic approach to Web service QoS quantification that is based upon realistic data and representative data samples.

We have also demonstrated how the RESTful style multimedia Web service achieved better improvement in gain and performance compared to SOAP.

The proposed approach can be extended and adapted to accommodate the specific criteria imposed by LAN/WAN, wireless LAN, Ad-Hoc, mobile networks, IP-Telephony and distributed systems.

5.8.1 Future work

We plan to extend our approach to include more QoS parameters, more multimedia types such as video and voice.

We would propose some recommendations, based upon our analysis results, in order to improve the QoS and performance of SOAP and RESTful multimedia Web services.

CHAPITRE 6

OPTIMISATION DE COMPRESSION D'IMAGES

Avant-propos

Auteurs et affiliation :

AL-CANAAN Amer ; étudiant au doctorat et chargé de cours au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

KHOUMSI Ahmed ; professeur au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

Date d'acceptation :

16 mai 2011.

État de l'acceptation :

version finale publiée.

Revue :

First International Conference on Wireless Communications and Mobile Computing (MIC-WCMC 2011).

Référence :

A. Al-Canaan and A. Khoumsi, «Performance Enhancement Of Image-Retrieval Web Services Through Image Dimensional Optimisation ». In Proceedings of Mosharaka

International Conference on Wireless Communications and Mobile Computing (MIC-WCMC2011), Istanbul, Turkey : 3-5 June 2011, pp.45-50.

Titre français :

Amélioration de la performance de services Web de récupération d'images par une optimisation dimensionnelle

Contribution au document :

Cet article correspond à la contribution 4 indiquée à la section 1.4.1. Celle-ci consiste à analyser et étudier la corrélation entre la dimension d'une image et le taux de compression, en utilisant des outils comme JPEG, JEPG2000 et DjVu. À souligner que l'originalité de cette contribution est dû au fait qu'elle propose une nouvelle piste de recherche qui n'a pas été abordée auparavant par d'autres recherches connexes dans ce domaine. Les résultats de notre étude montrent qu'il est pertinent d'utiliser des images dont le ratio de la grandeur sur largeur respecte le ratio d'or, $\frac{1+\sqrt{5}}{2}$, pour augmenter le taux de compression. Ces résultats sont confirmés pour des taux de compression élevés et modérés, c'est-à-dire supérieurs à 0.5.

Résumé français :

La performance de services Web multimédia de récupération d'images peut être améliorée en réduisant la taille du contenu des messages échangés, en particulier, les tailles d'images. Ceci peut être accompli de quelques manières telle que la compression d'images. La compression en soi, peut-être optimisée d'avantage. Dans cet article, nous étudions l'influence des dimensions de l'images, plus précisément, l'impact de la dimension sur le taux de compression. L'objectif est d'exploiter la corrélation entre le taux de compression et les ratios de dimensions des images (en anglais, *aspect-ratio*), le rapport de la longueur sur la largeur. Ceci permettra de changer les dimensions d'une image pour améliorer son taux de compression. Par conséquent, ceci augmente la performance de services Web de récupération d'images.

Mots clés : services Web, récupération d'images, performance, QdS, compression d'images, *aspect-ratio*, nombre d'or, DjVu, JPEG.

Note :

À la suite des corrections demandées par les membres du jury, le contenu de cet article diffère de celui qui a été soumis.

6.1 Abstract

The performance of image-retrieval Web services can be enhanced through the reduction of payload size, in particular, image sizes. This can be achieved by several ways such as image compression. Compression itself can be optimised further. In this article we study the influence of image dimensions, more specifically, the impact of the aspect ratio on the compression ratio. The goal is to benefit from the correlation between image compression ratio and image aspect ratio (width to height). This allows reshaping images in order to enhance their compression ratio. This in turn increases the performance of image-retrieval Web service and their QoS.

Index terms— Web services, image-retrieval, performance, QoS, image compression, aspect ratio, golden ratio, DjVu, JPEG.

6.2 Introduction

The widespread of digital image content has increased the demand on image retrieval and compression algorithms for Web services. The image compression is widely used in many domains such as Web services, IP-Telephony, sensor networks and distributed systems.

Performance of image-retrieval Web services can be enhanced through image compression. Image compression itself depends on several factors. Several performance parameters such as response time, server delay, rendering time, download time, image-retrieval time and compression ratio [16] can be optimised.

In this paper, we study the impact of the aspect ratio ($width\ to\ height$, or $w\colon h$) on the compression ratio of JPEG (*Joint Photographic Experts Group*) images. This can help optimise the image compression and augment performance and QoS [15, 17]. Little or no research has been conducted on this field.

We developed a simulation program using Java (not shown here), which is to re-dimension, compress and analyse JPEG images explaining the correlation between the width and height of an image and its compression ratio. Three different compression algorithms are considered in this study: JPEG, JPEG2000 [9, 10] and DjVu [57]. This is achieved through using the ImageMagick [74], JasPer [9] and DjVu libraries [45].

The simulation procedure is shown on Figure 6.1, which can be resumed as follows:

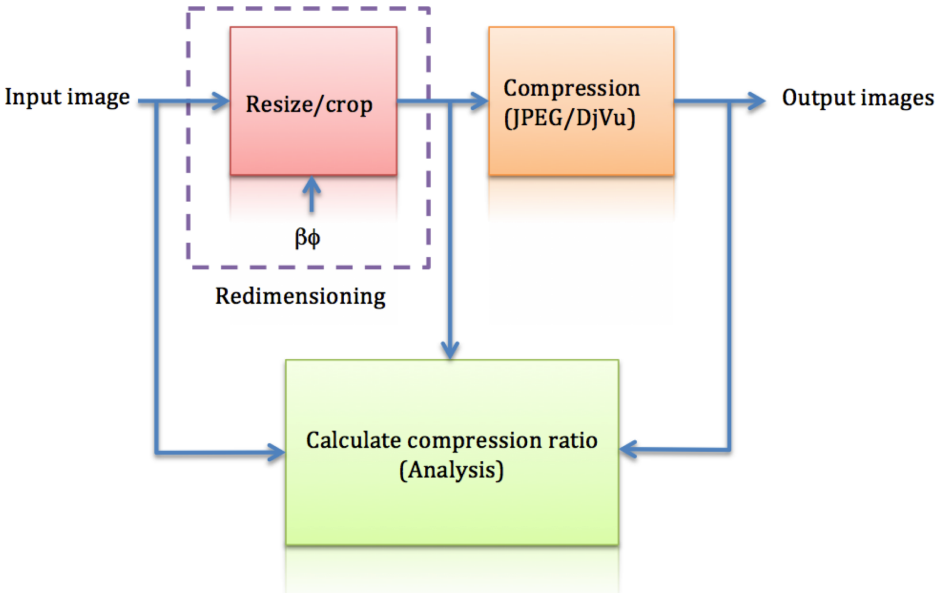


Figure 6.1 Block diagram of the simulation procedure.

1. Re-dimension image with deformation, i.e resize. Generally, for an input image of dimension $w \times h$, where $\{w,\, h\} \subset \mathbb{N}_{\geq 0}$, the new dimension of the resized output image is

$$n_1 w \times n_2 h,\ where\ \{n_1,\, n_2\} \subset \mathbb{R}_{\geq 0} \tag{6.1}$$

such that

$$\begin{cases} n_1 > 1,\ n_2 > 1 & \text{when re-sizing up} \\ n_1 < 1,\ n_2 < 1 & \text{when re-sizing down} \end{cases} \tag{6.2}$$

2. Re-dimension image without deformation, i.e image crop. Since cropping removes pixels from an image, equations (6.1) and (6.2) apply with $n_1 < 1,\ n_2 < 1$. The difference between re-sizing and cropping is that the latter removes information from an image, while re-sizing enlarges or minimises objects.

3. Compress the resulting image utilising JPEG, JPEG2000 and DjVu compression schemes.

4. Analyse collected results. A certain Java program iterates on the input images and calculates the compression ratios for each resized image.

The main contribution of this paper is the utilisation of the correlation between the aspect ratio of an image, the golden ratio and the compression ratio in order to attain better compression ratios. We propose a new aspect ratio for images, based upon the natural golden ratio, where high performance and compression ratio is desired. This is of high importance of image-retrieval Web services, IP-Telephony and sensor networks.

The sections of this paper are organised as follows: The related work is in section 6.3. The JPEG, JPEG2000 and DjVu image compression tools are discussed in section 6.4. Section 6.5 presents the re-sizing scheme. Section 6.6 presents the analysis method and the results of the analysis. Finally, the conclusion and future work are in sections 6.7 and 6.7.1, respectively.

6.3 Related work

The paper in [84] proposes a scheme to compress images in one dimension, which means that the images have to be stored in one vector or one-dimensional array before being compressed. Three methods are considered including wavelets, piece-wise approximation with triggers or PAT, and a modified version of PAT. The proposed scheme is less complex and thus suited for low-resource portable devices.

The work in [122] is motivated by the space complexity of PCA (*Principal Component Analysis*) and the spatial information loss due to matrix-to-vector representation of images for face-image databases.

Little or no work has discussed the correlation between the aspect ratio of an image and the compression ratio.

6.4 Image compression

Lossy JPEG is a standard image compression technique, which is based upon the DCT [117], while JPEG2000 [10] is based upon the wavelets transform[1] [22]. To enable

[1] Usually, the file extensions .jpg and .jp2 are used for JPEG and JPEG2000 standards, respectively.

ImageMagick to handle images using the JPEG2000 standard, the JasPer [9] library has to be installed, which can be accessed from within Imagemagick or directly from the command line.

On the other hand, DjVu is a different compression scheme, which produces images that are 5 to 10 times smaller than what JPEG can offer (in the case of a page of a colour magazine scanned at 300 dpi) [57]. DjVu classifies the components of an image before compressing them differently. The classification includes foreground (text and drawing) and background (pictures, paper colour and texture). For JPEG compression, we utilised the quality option with 70% and 50%, which means 30% and 50% compression ratios, respectively. For example, the following command of the ImageMagick library compresses an image with quality of 70%, i.e 30% compression ratio:

```
convert img_in -quality 70 img_out.jpg
```

The JPEG2000 compression ratio utilised is $100 : 1$, which is achieved by the following command for the input image and output images, `img1` and `img2`, respectively:

```
jasper -f img1 -T jp2 -F img2 -O rate=.01
```

The DjVu library command *c44* provides a reasonable compression ratio in DjVu format. The quality of compression ratio is determined by several quality options, such as $-slice\ 74 + 15 + 10$, which is interpreted as the number of slices per chunk: 3 chunks with 74, 15 and 10 slices, respectively. The number of chunks can be 1 up to 4 and the number of slices can be up to 120. The *c44* command with default values is as follows:

```
c44 image_in.jpg image_out.djvu
```

6.5 Image manipulation

A variety of input images has been selected for the analysis and for the experiments:

Group 1 A total of 101 images ranging from $13.5KB$ up to $4.873MB$ (122×160 pixels up to 5184×3456). Within this group, other subgroups have been created such as the 3648×2736.

Group 2 A total of 99 images of the same dimensions 2816×2112, i.e same aspect ratio.

Group 3 A total of 3 standard test images of identical dimensions 512×512, i.e $1 : 1$ aspect ratio [1], for this facilitates the comparison with other related works.

6.5.1 Re-sizing images

When an image is resized it appears larger or smaller in one or both dimensions, w and h. When enlarging an image, some kind of interpolation is usually applied to produce new pixels. On the other hand, reducing the size of an image involves sub-sampling to reduce the number of pixels.

For simplicity, we change either the width or height of an image (w or h). Changing either w or h at a time in equation (6.1) results in the following equation:

$$n = \frac{w}{h} \Rightarrow w = nh, \quad \text{where } n \in \mathbb{R}_{\geq 0} \tag{6.3}$$

In order to keep the deformation moderate we substitute n in equation (6.1) by the compound scalar value $\beta\phi$, where ϕ is the golden ratio (1.61803399) [81] and β is a variable such that $\beta \in [0.2,\ 2.0]$.

$$\beta\phi = \begin{cases} w/h, & \text{if } w > h \\ h/w, & \text{if } w < h \end{cases} \tag{6.4}$$

For an input image with $w > h$ (as shown on figure 6.2), from equation (6.3), the resized image is

$$[w \times h] \rightarrow [w \times \frac{w}{\beta\phi}] \tag{6.5}$$

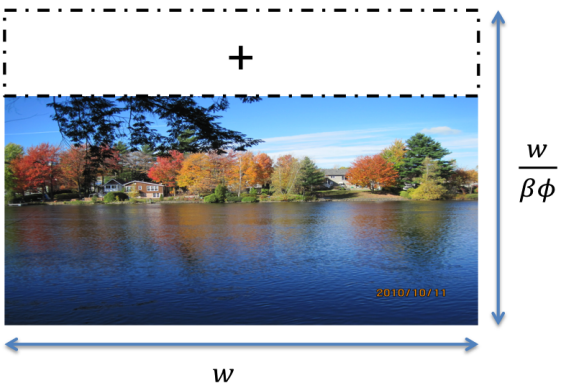
Figure 6.3 represents a golden rectangle, which is dissected into a square and a smaller rectangle. The following equations derive the calculation of the golden ratio ϕ as well as its conjugate $\phi^{'}$:

$$\frac{\phi}{1} = \frac{1}{\phi - 1} \Rightarrow \phi(\phi - 1) = 1 \tag{6.6}$$

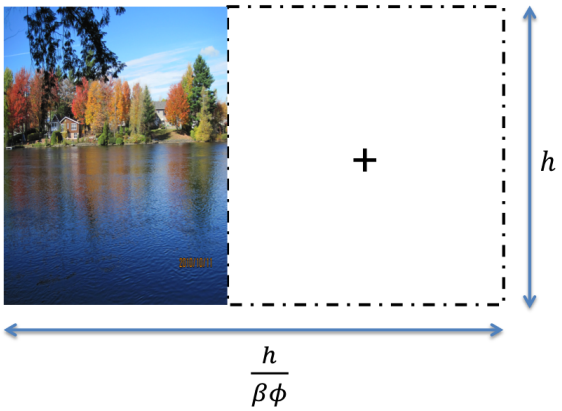
$$\phi^2 - \phi - 1 = 0 \quad \text{(2 solutions)} \tag{6.7}$$

$$\phi = \frac{1 + \sqrt{5}}{2}, \phi^{'} = \frac{1 - \sqrt{5}}{2} = \frac{-1}{\phi} \tag{6.8}$$

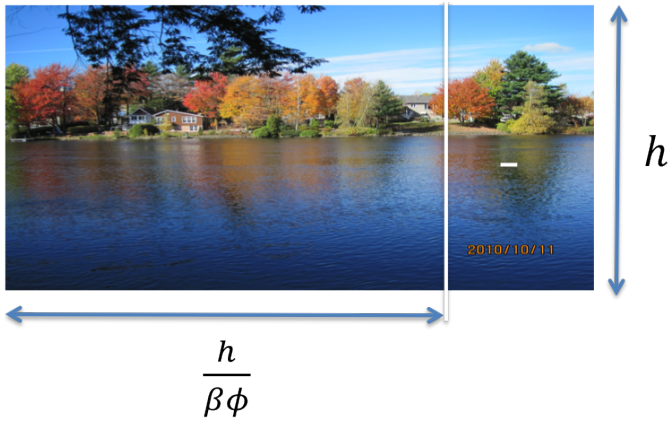
$$\phi = 1 - \phi^{'} = 1 - \frac{-1}{\phi} = 1 + \frac{1}{\phi} = 1 + |\phi^{'}| \tag{6.9}$$



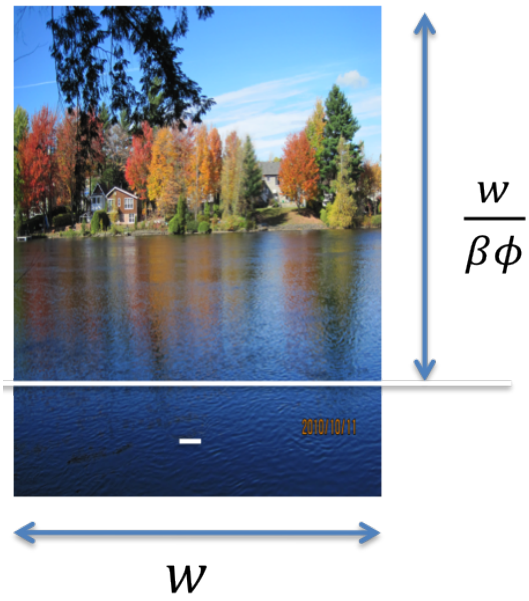
(a) Re-sizing an image with $w > h$.



(b) Re-sizing an image with $w < h$.



(c) Cropping an image, $w > h$.



(d) Cropping image, $w < h$.

Figure 6.2 Re-dimensionning images through cropping and re-sizing.

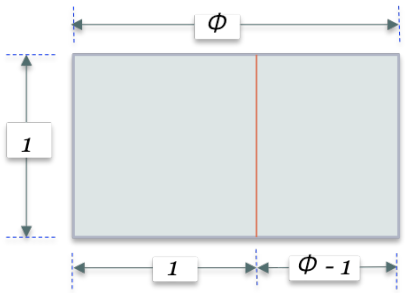


Figure 6.3 Golden rectangle to derive the value of ϕ.

6.5.2 Cropping images

Cropping means removing the outer pixels of an image in order to reduce its size. The aspect ratio of an image ca be modified by cropping the image in one dimension, i.e, by reducing either w or h. Unlike re-sizing, cropping could eliminate useful information. The dimensions of a cropped image with $w < h$, as shown on figure 6.2, can be calculated using the equation in (6.5) such that the cropped image must have less or equal height as follows:

$$\frac{w}{\beta\phi} \leq h \;\Rightarrow\; \frac{w}{h} \leq \beta\phi \text{ , where } \frac{w}{h} \text{ is the aspect ratio}$$

Similarly, for an input image with $w > h$, the resulting cropped image satisfies the following relation:

$$\frac{h}{\beta\phi} \leq w \;\Rightarrow\; \frac{w}{h} \geq \beta\phi \text{ , where } \frac{w}{h} \text{ is the aspect ratio}$$

6.6 Experimental results

In this section, we analyse the obtained results. The influence of β, and thus the aspect ratio, on the compression ratio for the sample group 1 is shown on figure 6.4. As can be noticed on the figure, the compression ratios for both DjVu and JPEG of quality 70% (30% compression) are decreasing as β increases. The increasing non-linear cure represents the change image size as β increases. The negative values shows that the resized images have larger size than the original. Obviously, the two points $\beta = 0.612$ and $\beta = 1.0$ offer a good compromise between the aspect ratio and compression ratios for both DjVu and JPEG. For low compression ratios between 10% and 40%, the point $\beta = 0.612$ offers good compromise between image compression ratio and aspect ratio. On the other hand, for higher compression ratios greater than 40%, the optimal point becomes at $\beta = 1.0$, since it produces higher reduction in the image size.

On figure 6.5, the images in group 1 have one dimension, i.e, 3648×2736. The perpendicular line represents the corresponding value of $beta$ for the original aspect ratio, which is equal to $3648/2736 = 1.33333$. The JPEG compression at 50% quality approaches the 50% compression value. For $\beta = 1$, the image size decreases by 6% and the compression ratios decrease by less than 2%. This indicates that at $\beta = 1$ there is a good compromise between the image compression ratios and aspect ratio.

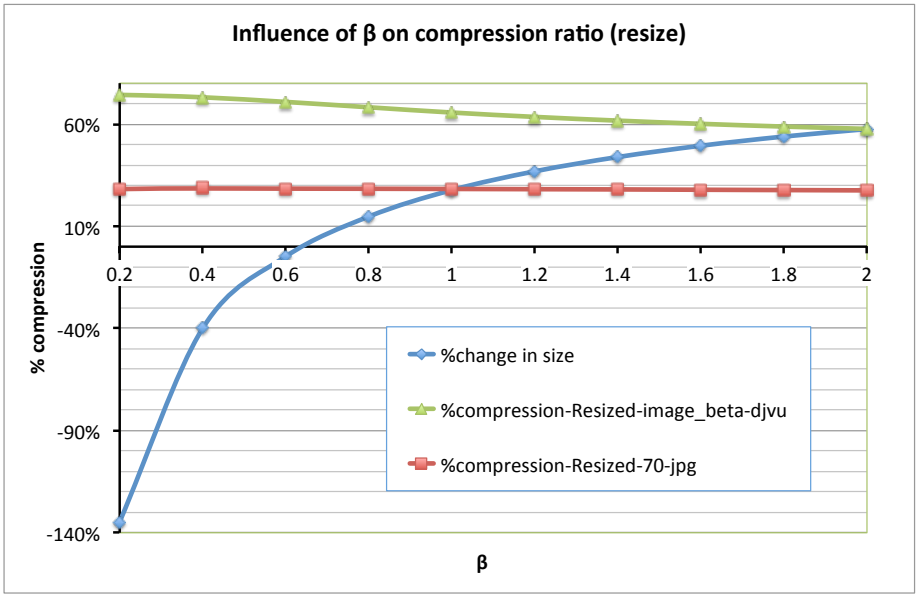


Figure 6.4 Influence of β on image compression (resize, quality 70%).

For $\beta = |\frac{1}{\phi}| = 0.612$, the compression ratios become a little bit higher at the expense of increasing the image size compared tot the original size. For this reason, the point at $\beta = 1$ offers a better compromise, specially when high compression ratios are desired, i.e, above 50%, as shown on figure 6.5.

On the other hand, for $\beta > 1$, the compression ratios become less interesting, since the curves of the compression ratios and the change in image size converge and approach each other. This makes compression inefficient. We consider the compression effective if the resulting size is at least 10% less than that of the original image.

Figure 6.6 shows that the cropping operation reduces the image size significantly and that both compression schemes become inefficient for $\beta \geq 1.2$. The optimum point for DjVu compression is close to $\beta = 1$, where the compression ratio of DjVu library is above the decrease in image size by 6.55%. However, the DjVu curve is increasing slightly and JPEG compression is slightly decreasing. In figure 6.7, both DjVu and JPEG at 50% quality are close to each other and obviously, the optimum point is for $\beta = 1$. The re-sizing scheme utilised is cropping. When re-sizing the images, as shown on figure 6.8, the optimum point is for $\beta = 1$. The three types of compression are high. The JPEG2000 curve tends to stabilise for $\beta \geq 1$, while DjVu and JPEG curves decrease.

When cropping is utilised, as shown on figure 6.9, the optimum point is at $\beta = 1$ for the 3 compression schemes. At $\beta = 0.612$ a similar optimal point is noticed specially when the compression ratios are below 70%.

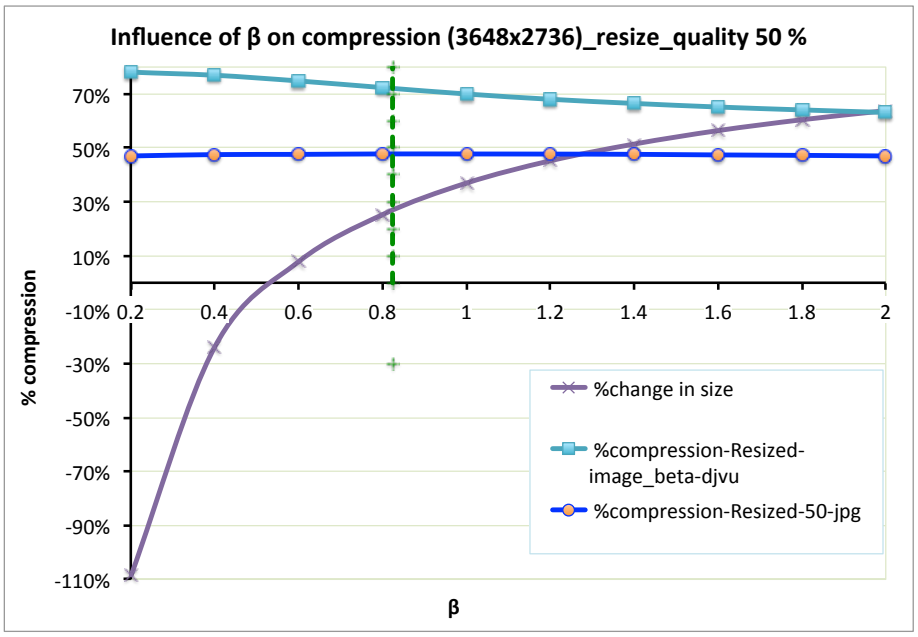


Figure 6.5 Influence of β on image compression (resize, quality 50%).

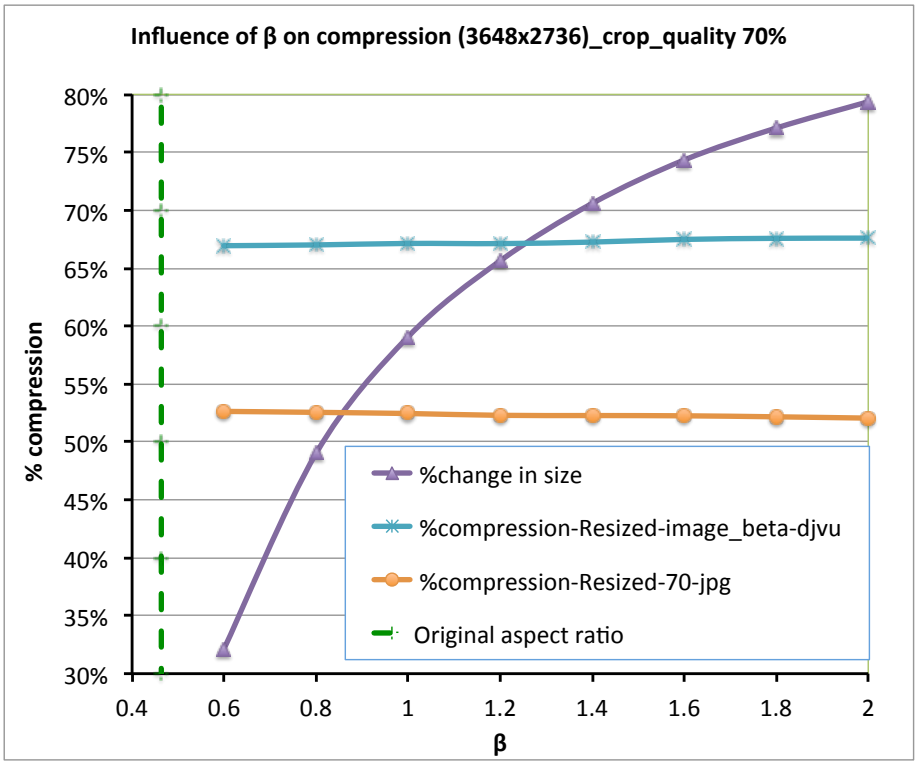


Figure 6.6 Influence of β on compression ratio (crop, quality 70%).

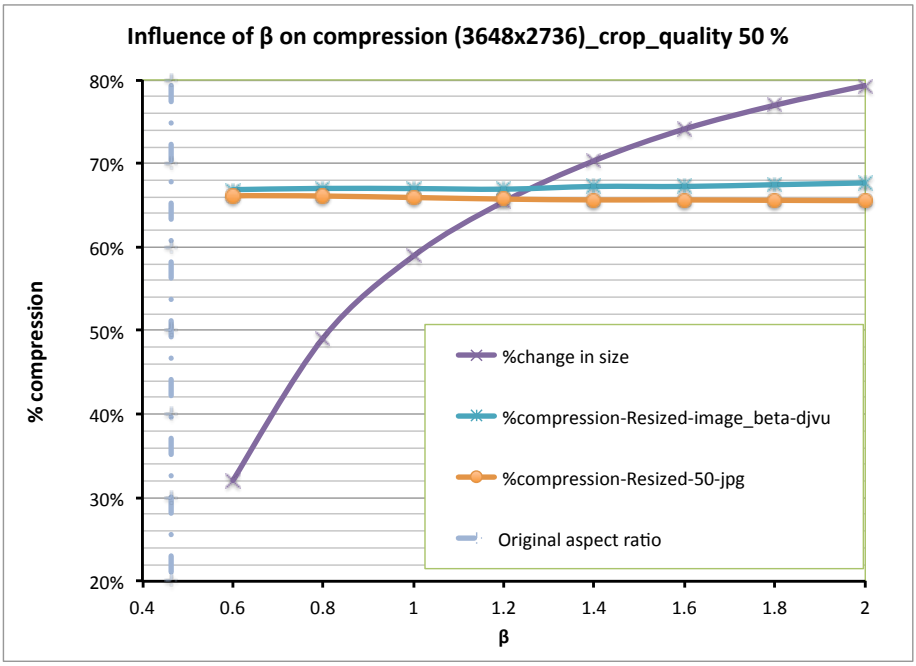


Figure 6.7 Influence of β on compression ratio (crop, quality 50%).

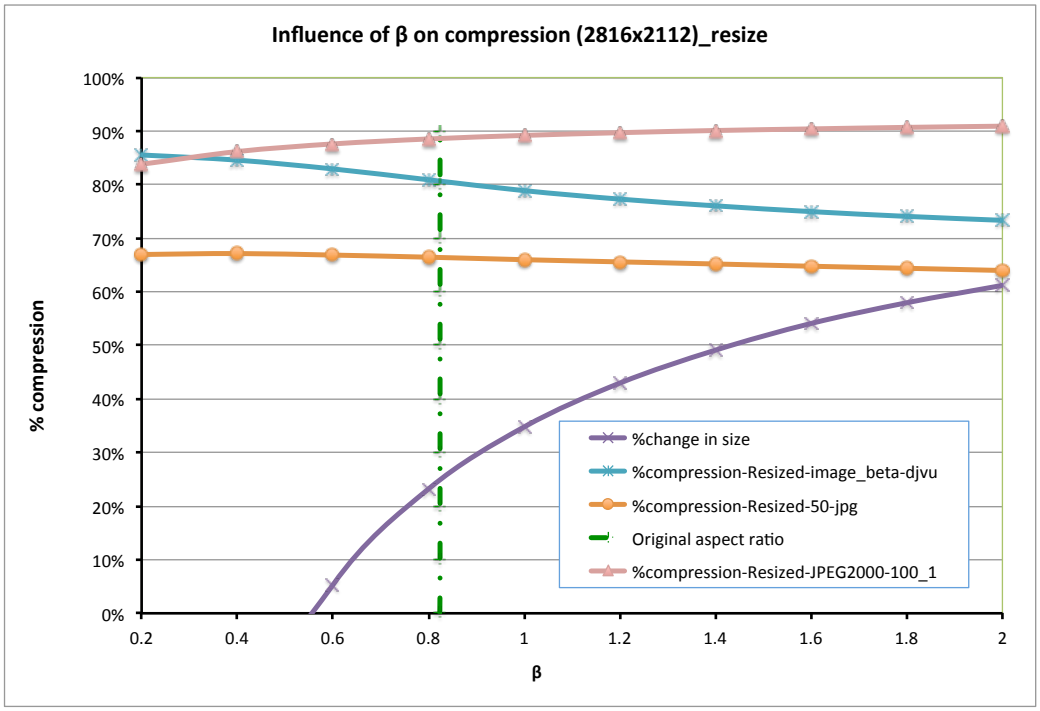


Figure 6.8 Influence of β on compression ratio (resize, quality 50%, 100:1).

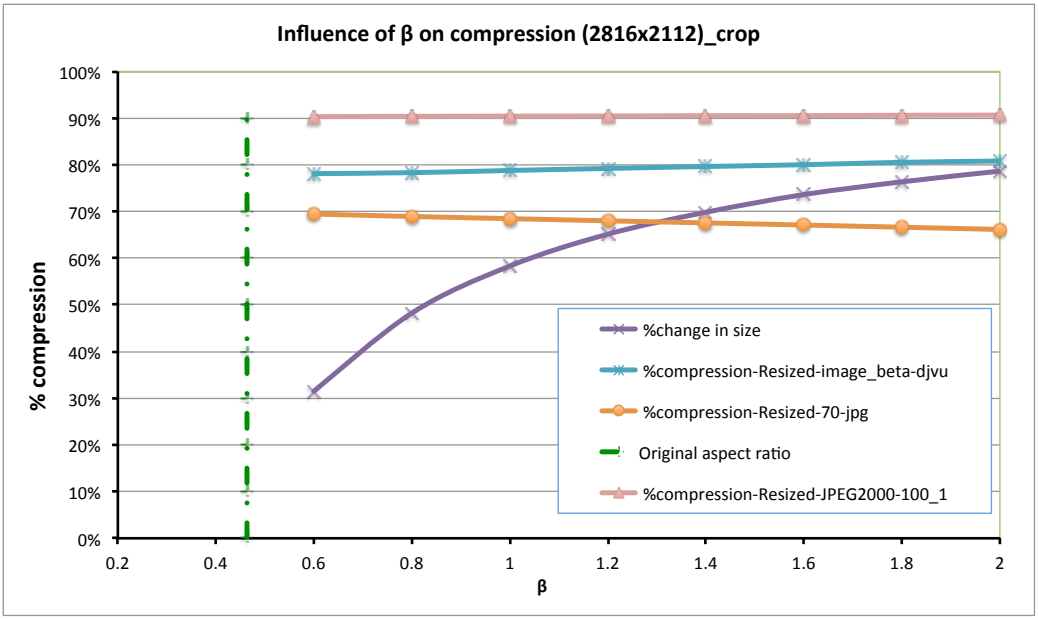


Figure 6.9 Influence of β on compression ratio (crop, quality 50%, 100:1).

6.6.1 Frequency-aware analysis for image compression ratio optimisation utilising the golden ratio

A set of 3 standard images [1] (originally in TTF[2] format) are utilised in the analysis of optimisation of compression ratio with frequency considerations. The dimensions and file size for each of the 3 TFF images are 512×512 pixels and $786.572\ K$ bytes, respectively. The original images have been first converted to JPEG format with the following compression ratios: 44.94% for the image on figure 6.10a, 51.67% for the image on figure 6.10d, and 30.38% for the image on figure 6.10g).

The aspect ratio of the 3 square-shaped test images is $1:1$, for which $\beta = 0.612$. This value of β is specially interesting, since it is equal to $|\phi'|$, i.e, is the magnitude of the golden number conjugate calculated in equation 6.8.

The differences in compression ratio between the 3 images are attributed to the texture profile and distribution of details and colours in each image. The image shown on figure 6.10d has the least variations in colours and details, therefore its compression ratio is the highest, i.e, 51.67%. This is confirmed by the lowest new JPEG size of $380.176\ K$bytes as shown on Table 6.1. The high frequency image on figure 6.10i reveals

[2]TTF: Tagged Image Format can be utilised as a container of JPEG (lossy compression) as well as any lossless compression scheme.

more details than the image on figures 6.10c, which explains the lower compression ratio for the image 3.

The low and high frequency image versions, as shown on figure 6.10, have been produced as follows:

1. apply a Gaussian blurring low-pass filter to the original images (blurred image).
2. subtract the low pass filtered images from the original images (subtracted image).
3. blend each blurred image (in step 1) with the original image in order to produce the low frequency image versions (as shown on figures 6.10b, 6.10b and 6.10h).
4. blend each subtracted image (in step 2) with the original image in order to produce the high frequency image versions (as shown on figures 6.10c, 6.10c and 6.10i)[3].

After applying the procedure explained on figure 6.1 in the previous section 6.2, the collected data are analysed for every iteration, such as the first iteration ($\beta = 0.2$) as shown in Table 6.1.

The first row of images on figure 6.11 represent the same test procedure as in the previous section 6.6 applied to the 3 input images of the same dimension and image size. For the images on figures 6.11a and 6.11b, the point at $\beta = 1$ is a good compromise for all compression schemes utilised, since the deformation is away from the severe region and the modification in size leads to a reduction in size. The effective compression ratios for both configurations are above 50%. On the other hand, for lower compression ratios (between 20% and 50%), the point at $\beta = 0.612$ becomes more interesting, i.e, at the original dimension (view figure 6.11b). Another remark on both figures of the first row is that the reduction in image size is almost the same for both operations: crop and re size (shrink or extend).

The second row on figure 6.11 shows the effect of low and high frequency on the analysis. The point at $\beta = 1$ is a good compromise when compression ratio is above 30% for the low frequency versions, as shown on figure 6.11c. From the same figure, we conclude that the point at $\beta = 0.612$ is only advantageous when the compression ratios are between 10% and 30%. The point at $\beta = 1$ is a good compromise when compression ratio is above 50% for the high frequency versions, as shown on figure 6.11d. From the same figure, we conclude that the point at $\beta = 0.612$ is only advantageous when the compression ratios are between 20% and 50%.

[3] All of the previous steps are generated by a script based upon the ImageMagick library [2].

(a) Original image 1.

(b) Low frequency version.

(c) High frequency version.

(d) Original image 2.

(e) Low frequency version.

(f) High frequency version.

(g) Original image 3.

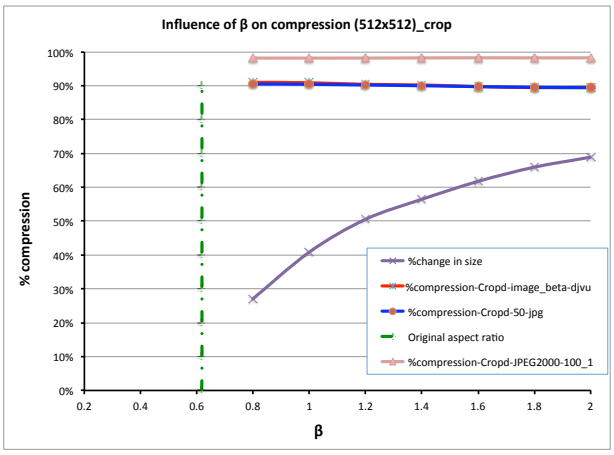
(h) Low frequency version.

(i) High frequency version.

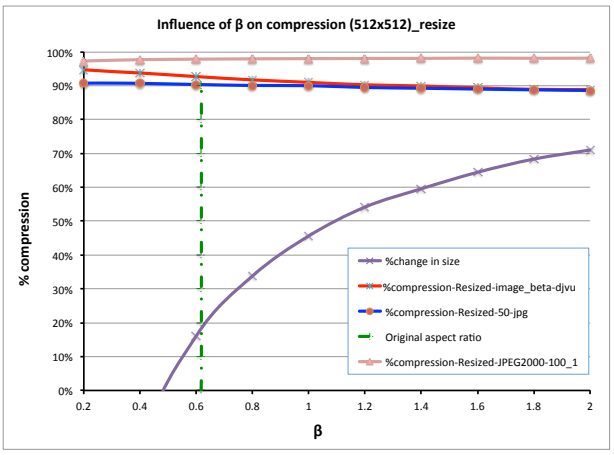
Figure 6.10 Standard test images utilised in the frequency-aware analysis.

Table 6.1 Iteration when $\beta = 0.2$ of the analysis of compression ratio (resize, quality 50%, 100:1).

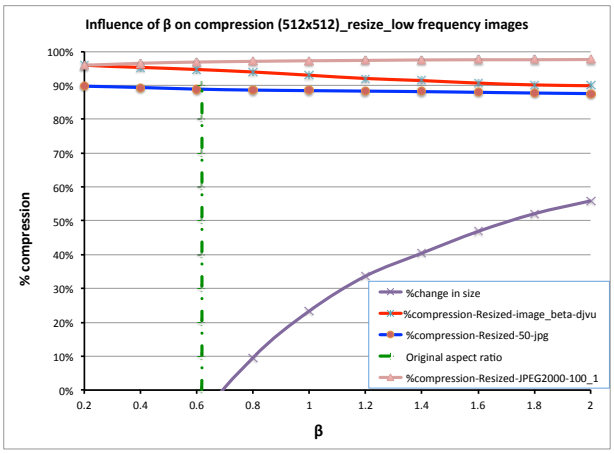
512x512							Resize dimension 1582x512								
	Original	Original					β = 0.2	DjVu	JPG	JPG_50%	JPG2000	%DjVu	%JPG_50	%JPG_2000	%Change_size
Image name	Dimensions	Size (Kbytes)	JPG70%	DjVu	%JPG70	%DjVu	new dimensions	Kbytes	Kbytes	Kbytes	Kbytes	0.2	0.2	0.2	0.2
house.jpg	512x512	433.069	57.125	38.576	**86.8%**	**91.1%**	1582x512	53.458	**881.531**	89.489	24.274	93.9%	89.8%	97.2%	-104%
4.2.06.jpg	512x512	547.648	62.147	45.875	**88.7%**	**91.6%**	1582x512	58.868	**1093.634**	96.18	24.295	94.6%	91.2%	97.8%	-100%
4.2.05.jpg	512x512	380.176	43.681	25.931	**88.5%**	**93.2%**	1582x512	37.011	**780.253**	65.862	24.279	95.3%	91.6%	96.9%	-105%
Average		453.631	54.31767	36.794	87.99%	91.96%		**49.779**	**918.4727**	**83.84367**	**24.282667**	**94.60%**	**90.87%**	**97.30%**	**-102.83%**



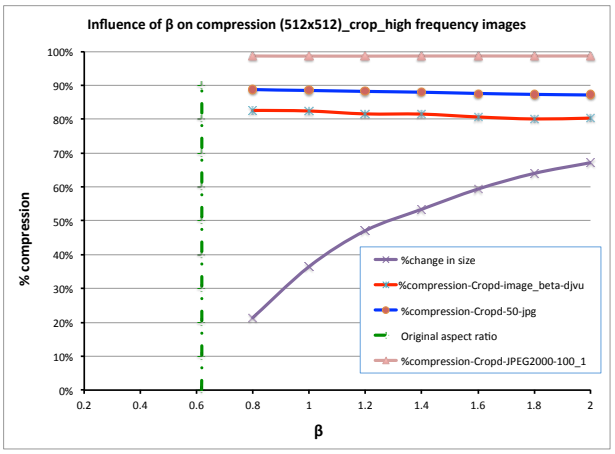
(a) Effect of crop



(b) Effect of re size



(c) Effect of low frequency resized images.



(d) Effect of high frequency resized images

Figure 6.11 Compression ratios versus β for low and high frequency test images.

6.7 Conclusion

The correlation between an image's aspect ratio and its compression ratio allows better compression ratio, based upon DjVu, JPEG and JPEG2000 compression techniques, and optimised performance for image-retrieval Web services.

When moderate, above 50%, and high image compression ratios are desired ($\geq 50\%$), we propose ϕ (golden ratio) as the aspect ratio, which provides a good compromise between input file size and compression ratio.

For low compression ratios, below 30%, another optimal point may exist when the aspect ratio is $1:1$. This is also related to the golden ratio, since $1 = \phi\,|\frac{1}{\phi}| = \phi\,|\phi'|$. Based upon the analysis results we obtained, we propose the golden ratio as the aspect ratio for images to be used in multimedia Web services, in order to optimise the compression ratio.

The dimensions of an image have an influence on its compression ratio. In the case of DjVu compression scheme, the correlation is more obvious compared to JPEG and JPEG2000. The size of an image decreases as β increased.

The frequency-aware analysis revealed that the standard JPEG compression technique is the same for low and high frequency images with different β values, while the JPEG2000 and DjVu have different behaviours. The point at $\beta = 1$ is still a good compromise for the 3 compression techniques. The optimal zone around $\beta = 1$ is valid for high compression ratios above 50% for the high frequency images and above 30% for the low frequency images.

6.7.1 Future work

We plan to extend our approach and results in order to apply the recommended aspect ratio into different compression schemes for image and video contents.

We would propose some recommendations, based upon our analysis results, in order to improve the QoS and performance of multimedia Web services including voice and video.

CHAPITRE 7

MISE EN ŒUVRE DE SERVICES WEB MULTIMÉDIA SUR FPGA

Avant-propos

Auteurs et affiliation :

AL-CANAAN Amer ; étudiant au doctorat et chargé de cours au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

KHOUMSI Ahmed ; professeur au département de génie électrique et de génie informatique, Faculté de génie, Université de Sherbrooke.

Date de soumission :

29 juillet 2014.

Revue :

International Journal of Web and Grid Services.

Titre français :

Vers une architecture d'une plate-forme polyvalente de services Web multimédia RESTful à haute performance mise en œuvre sur FPGA.

Titre anglais :

«Towards a Comprehensive Platform Architecture for High Performance RESTful Multimedia Web Services Implemented on FPGA ».

Contribution au document :

Cet article correspond aux contributions 5 et 6 indiquées à la section 1.4 qui se résument comme suit :

- Élaborer une méthode de conception et de mise en œuvre cohérente qui se concentre sur la création de SWM à haute performance et à haute qualité de service (QdS).
- Offrir une liste de lignes directrices, des astuces et des outils pertinents à prendre en compte lors de la conception de SWM sur FPGA. Ceci a pour but de faciliter les étapes de conception et de mise en œuvre sur FPGA, d'aider les concepteurs à raccourcir le temps de conception et de prendre des décisions plus rapidement.
- Quantifier la QdS des SWM mis en œuvre sur FPGA et comparer nos résultats obtenus avec d'autres travaux de recherches connexes dans ce domaine.
- Offrir une mise à jour des documentations et du code HDL (pour le projet TCP/IP socket utilisé dans la section 7.6.1) à source-ouverte que nous avons adapté pour la conception de SWM sur FPGA incluant des modules de communication Ethernet entre le FPGA et le Web. Les documentations mises à jour dans les annexes A et B sont liées au module Ethernet IP Core (utilisé dans le deuxième exemple de conception et de mise en œuvre sur FPGA dans la section 7.6).

Résumé français :

La disponibilité répandue de cartes de développement FPGA qui sont abordables encourage l'utilisation de FPGAs dans des domaines multiples, tels que le traitement accéléré dédié de données ainsi que des applications Web. Comparativement aux ASICs, les FPGAs présentent l'avantage de réduire le temps entre la conception et la mise en marché d'un produit fini fonctionnel. Comme les ASICs, les FPGAs incluent le parallélisme permettant le traitement d'algorithmes complexes et en temps réel. L'utilisation de FPGAs est favorable pour les applications de systèmes embarqués et mobiles. Bien que les solutions fondées sur les ASICs soient plus avantageuses que celles de FPGAs, avec une plus grande vitesse et une taille plus réduite, elles nécessitent

des coûts et du temps de conception et de mise en œuvre considérables, ce qui les rend moins populaires dans plusieurs domaines.

Les services Web sur FPGAs sont de plus en plus communs avec la disponibilité d'outils de conception et de code source ouvert. Plusieurs réalisations de services Web multimédia mettent l'accent sur des services légers ayant des tailles modestes de données. Certaines réalisations sont fondées sur des couches intermédiaires afin d'établir des services Web sur FPGA en exploitant un microprocesseur et une version particulière d'un noyau Linux. Alors que ceci pourrait faciliter le développement sur FPGA en profitant des librairies déjà existantes pour la communication entre FPGA et le Web à travers un port Ethernet, il réduit considérablement la performance et l'efficacité.

Dans cet article, nous fournissons deux exemples de conception de services Web multimédia (SWM) sur FPGA. Le premier vise l'élaboration de quelques lignes directrices de conception sur FPGA incluant quelques outils logiciels. Le deuxième exemple est concentré sur la conception d'un service Web multimédia sur FPGA Spartan6 à haute performance et à haute qualité de service (QdS), c'est-à-dire que ce dernier met l'emphase sur les couches supérieures du modèle OSI. L'avantage de la solution proposée est qu'elle permet d'exploiter au maximum des ressources matérielles sur FPGA.

Mots clés : services Web, FPGA, REST, performance, QoS, temps de réponse, délai du serveur, VHDL, Verilog.

Note :

À la suite des corrections demandées par les membres du jury, le contenu de cet article diffère de celui qui a été soumis.

7.1 Abstract

The widespread of FPGA development boards, which are available at moderate prices, promotes the utilisation of FPGAs for several applications, such as dedicated accelerated processing and Web applications. Compared to ASICs, FPGAs present the advantage to reduce the design-to-market time that is needed to obtain a fully functional product. FPGAs (as well as ASICs) include parallelism allowing the processing of complex and real-time algorithms. The utilisation of FPGAs is compelling for embedded systems

and mobile applications. Although ASIC-based solutions have some advantages over FPGAs, such as higher speed and smaller dimensions, they necessitate a much more considerable cost and time of design and implementation, making them less popular in many domains.

Web services on FPGAs are becoming common with the popular design tools and open-source resources. Several realisations of multimedia Web services focus on lightweight services with modest sizes of data. Some realisations are based upon intermediary layers to achieve Web services on FPGA by implementing a microprocessor and a flavour of Linux kernel. While this may facilitate the development on FPGA by exploiting the already available libraries for communication between the FPGA and the Web through the Ethernet port, it reduces significantly performance and efficiency.

In this paper, we provide two design examples of multimedia Web service (MWS) on FPGA, the first of which aims at elaborating several design guidelines on FPGA including several software tools. The second example concentrates on high performance and high quality of service (QoS) design of MWS on Spartan6 FPGA, i.e., with more emphasis on the upper layers of the OSI model. The advantage of the proposed solution is that it allows maximal exploitation of hardware resources on FPGA.

Index terms— Web services, FPGA, REST, performance, QoS, response time, server delay, VHDL, Verilog.

7.2 Introduction

7.2.1 FPGA compared to other technologies

FPGAs (*Field Programmable Arrays*) became widespread due to several characteristics including parallelism, low-power consumption, their ability to be reprogrammed, affordable price and availability of design tools. Low-power consumption is compelling for embedded systems and distributed applications. Many engineering domains took advantage from FPGAs, such as ASIC (*Application Specific Integrated Circuits*) prototyping and educational projects.

Compared to FPGAs, ASICs offer lower area on Silicon (around a 100 times less than FPGA), better performance (between 3 and 4 times faster) and lower power consumption

(around 14 times less) [69]. These figures may vary according to different design and applications constraints. On the other hand, FPGAs provide a shorter time-to-market, a low-cost and flexible platform for prototyping and educational purposes. The maximum usable frequency depends on the area occupied in the FPGA chip and on the peripherals around. A detailed comparison inspired by [40, 70] between ASICs, FPGAs, CPUs and GPUs is shown in Table 7.1, which shows that the choice of a technology is highly application-dependent. Usually a number of weighted factors determine the appropriate technology to be used. The positive score notation (denoted by the ✓sign) in Table 7.1 helps take a quick decision. For example, if parallelism and low cost are the main requirements, the recommended approach is FPGA since it has the highest score of 8. If low-power consumption and re-programmability are the main concerns, FPGA is obviously the dominant choice. In terms of speed, ASICs are reported to be three times up to eight times faster than FPGAs [70].

7.2.2 Motivation

Due to parallelism, the utilisation of FPGAs becomes relevant in accelerating various processing-intensive tasks, such as compression algorithms and multimedia manipulation. The hardware nature and flexibility of FPGAs make them appealing to Web services designers, for their inherent immunity against software viruses for instance. These advantages have motivated our research work through the utilisation of FPGAs in high quality, secure and high performance multimedia Web services (MWS). Different Web services were successfully designed on FPGAs without necessarily focusing on the Quality of Services (QoS). We therefore demonstrate the design and realisation of MWS on FPGA, while quantifying their QoS through a two-step approach. We provide a first example of design of a MWS that incorporates the addition of an Ethernet hardware module in addition to HDL modules. The objective of the example is to provide a list of design guidelines on FPGA that are beneficial to help accomplish designs of MWS on FPGA. The example also concentrates on the lower layers of the Open Systems Interconnection (OSI) model. We provide a second example of design of a MWS on FPGA that concentrates mainly on the higher layers of the OSI model, as well as on several QoS parameters. In our realisation, we use pure HDL code, open-source modules and free design tools.

Table 7.1 Comparison between ASIC, FPGA, CPU and GPU.

Criterion	ASIC	FPGA	CPU	GPU
High Speed[1]	✓✓✓✓	✓✓✓	✓✓	✓✓
Throughput	✓	✓	✓✓✓	✓✓✓✓
Parallelism	✓✓✓✓	✓✓✓✓	✓✓	✓✓
Multitasking[2]	✓	✓[3]	✓✓✓✓	✓✓
Low-power consumption	✓✓✓✓	✓✓✓	✓	✓
Re-programmable	✓	✓✓✓✓	✓✓✓✓	✓✓✓✓
Connectivity and interfacing	✓✓	✓✓✓✓	✓✓	✓
Low total cost[4]	✓	✓✓✓✓	✓	✓✓✓
Precise timing	✓✓✓✓	✓✓✓✓	✓✓	✓✓
Small dimension	✓✓✓✓	✓✓✓	✓✓	✓✓
Low design-to-market time	✓	✓✓✓	✓✓✓	✓✓✓
Data Security	✓✓✓✓	✓✓✓✓	✓	✓
Mobile applications	✓✓✓✓	✓✓✓✓	✓	✓
Embedded applications	✓✓✓✓	✓✓✓✓	✓✓✓	✓
Educational applications	✓	✓✓✓✓	✓✓✓✓	✓
Total score	40	47	34	30

[1] Considering the number of clock cycles needed to do an arithmetic operation.
[2] Considering the number of diverse processes that can be served.
[3] Multitasking can be achieved on an FPGA by the implementation of a CPU and a Linux kernel for example.
[4] Considering the cost of peripherals needed to obtain a functional project.

The REST (***Re****presentational* ***S****tate* ***T****ransfer*) style is proven to be more efficient and of lighter weight compared to SOAP-based multimedia Web services. Previous studies have shown the relevance of using REST style rather than the SOAP protocol for the design of MWS [16, 18, 27, 53, 91]. For designing MWS and managing their QoS and performances, we opted for the Nexys3 development board from Digilent [63], which is based upon the Xilinx Spartan6 [3] FPGA together with a few peripherals, notably a 10/100 Mbps Ethernet port and a Phy chip [46, 103].

7.2.3 Contributions

Our work can be distinguished from other works in several aspects. Firstly, we focus on designing and implementing high QoS and high performance multimedia Web services, where the payload data are in binary format (images, voice and video) rather than text and XML. We also cover up a wider range of data sizes, which are more realistic and common to multimedia Web services.

Secondly, we highlight and quantify the QoS of our designed MWS.

Thirdly, we highlight various details about our design and implementation tools and environments, which can be viewed as guidelines for choosing proper tools and methods for future designs and implementations.

Fourthly, we provide a comprehensive comparison between our results with FPGA versus other related works on multimedia Web services on FPGAs as well as on computers in order to analyse several relevant QoS parameters such as response time, download time and processing time.

The work in this paper can be extended to include specialised QoS management modules, enhancing the design and realisation of high performance and high fidelity MWS.

Our proposed architecture is completely realised with HDL without the addition of CPU layers on an FPGA in order to profit from the hardware resources on FPGA.

Throughout the design and implementation process of our MWS, we encountered different issues that we overcome by the adoption of different open-source HDL modules. This gave us the opportunity to contribute to the open-source community by updating different codes and documentations for some HDL modules and sharing knowledge with other developers and researchers.

7.2.4 Organisation

The sections of this paper are organised as follows: in section 7.3, we present an overview of related work. In section 7.4, we present a design example on FPGA. This example is used to deduce a list of design guidelines, which are presented in section 7.5 and are meant to help designers accomplish MWS design on FPGA. In section 7.6, we present a second example of design of a MWS on FPGA with a straightforward approach, where all necessary prerequisite hardware is present. In section 7.7, we discuss results and compare them to related works on MWS. Section 7.8 contains our conclusion and propositions of future work.

7.3 Related work

In this section, we first give an overview of related work on Web services' design on FPGA. The work in [97] aims at reducing implementation time and effort to create flexible and high performance Web services. The design is simplified through the utilisation of high level programming languages, which relies on an intermediary component on FPGA, namely the NIOS II micro-controller. This facilitates design by taking advantage of existing libraries.

The work in [40] presents a MWS for interfacing some specialised laboratory equipment on campus with remote end-users who may be equipped with PDA's. The realisation includes a Web server and FPGA Spartan-3E board. The design takes into consideration the low processing resources in PDA's, which aim at controlling the laboratory equipment and getting a visual feedback using a Web-cam. The paper focuses the attention on the usefulness of the features rather than the QoS and performance comparison.

The works in [28, 38] describe the use of an FPGA-based system to attack a crypto-processor deployed in a banking infrastructure based upon DES (*Data Encryption Standard*). The authors demonstrate the utilisation of pipe-lined solution and replication processing blocks on FPGA, as two valid and efficient schemes.

7.4 Design example on FPGA

This section presents an example of design of a MWS, video on-demand, based upon a UDP packet sender at 10 Mbps, which aims at highlighting a list of the design guidelines leading to seamless creation of MWS on FPGAs. This list of design guidelines will be recapitulated in section 7.6. Though the example in this section covers several layers of the OSI [103], we concentrate on the lower layers, i.e., the data link layer and the physical layer (Phy). This example also explains the design and realisation of an Ethernet hardware module in addition to HDL modules. An important advantage of this example, is that it exposes the I/O differential capabilities of FPGA and the possibility to integrate custom hardware interface to the FPGA card.

7.4.1 Design of a UDP and ARP packet sender on FPGA

Depending on the hardware requirements at the lower Phy layer and at the MAC (Media Access Control) sub layer[1], the designer may encounter at least one of three options as shown on figure 7.1. When the available peripherals do not include an Ethernet interface, a new Ethernet interface should be added, i.e., built or bought. In our example, we decided to build the Ethernet interface, since this is considered as a low-cost solution that also allows profiting from legacy FPGA hardware. This verification phase of available hardware prior to development corresponds to the design guideline 1 in section 7.5. For example, the abundance of free-of-charge D2SB [63] FPGA cards at campus justified our choice of building a custom 10-Mbps Ethernet extension card for the D2SB as depicted on figure 7.3. The FPGA's support for various differential I/O signalling standards is another motivation for our choice to build an Ethernet interface.

The features of the D2SB platform can be resumed as follows:

- Xilinx XC2S200E FPGA chip of the Spartan IIE family.
- A 50 MHz oscillator (non-crystal type) and a socket for a second oscillator.
- Six expansion connectors two of which can connect with the data I/O (input/output) board DIO5, providing 16 LEDs, a 4-digit 7-segment display, 16 push buttons, 8 switches and a two-line LCD display.
- ROM to store the generated programming files.

[1] The MAC is a sub layer in the Data link layer of the OSI model.

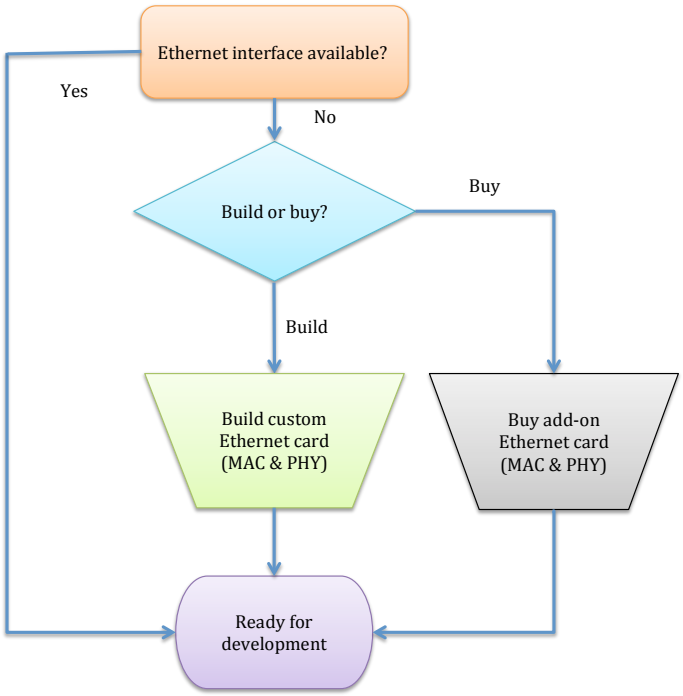


Figure 7.1 Flow chart of hardware availability procedure.

- More than a hundred I/O ports that are available at the six extension connectors.

The list of I/O standards supported by FPGA include LVTTL[2], LVCMOS2[3], LVDS[4] and LVPECL[5]. For low-speed applications with low interference and noise, LVTTL is sufficient, otherwise the utilisation of one out of several differential standards would be inevitable. Unlike the LVTTL and LVCMOS2 I/O standards, the differential signalling standard, on which depend many serial communications, such as Ethernet, necessitates a pair of wires per signal line. Ethernet Tx and Rx differential-pair signals can be fed directly into the FPGA by selecting matched differential I/O pairs of the D2SB as described in the data sheet. This leads to the design guidelines 2 and 3 of section 7.5.

Differential I/O pins offer best noise rejection and are carefully routed to obtain identical electrical characteristics on copper tracks, which means that the line widths, lengths and spacing should be identical. Our experiments on all of the differential I/O pairs of the D2SB revealed that almost all differential I/O pairs were properly matched except for one or two differential pairs.

[2] *Low Voltage Transistor Transistor Logic*
[3] *Low Voltage Complementary Metal Oxide Semiconductor*
[4] *Low Voltage Differential Signal*
[5] *Low Voltage Positive Emitter Coupled Logic*

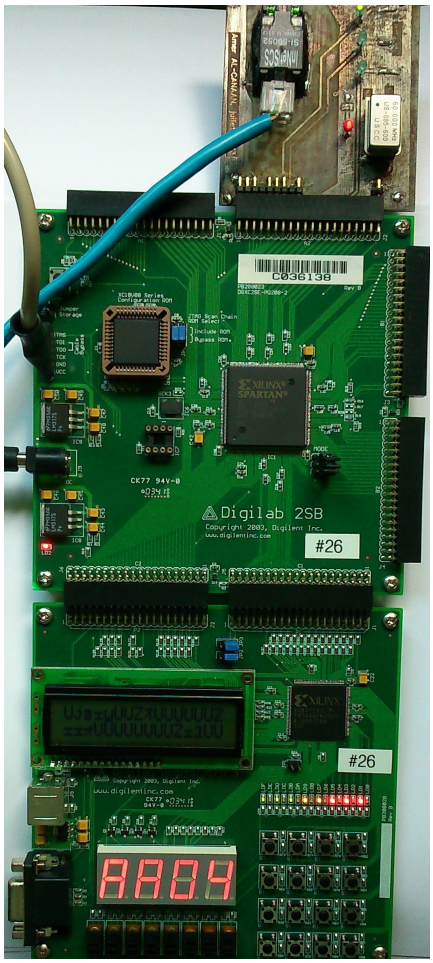


Figure 7.2 Custom Ethernet board connected with the D2SB board.

7.4.2 Overview of 10-Mbps Ethernet

Before proceeding to the implementation details we provide a quick overview of the Ethernet standard at 10 Mbps. This is important since the composition of Ethernet frames (at the MAC sub layer) and the generation of differential signals at the Phy layer are done by logic, through HDL code within the FPGA. For 10-Mbps Ethernet, Manchester encoding is utilised, where every bit of information is encoded as a transition from 1 to 0 or from 0 to 1. This is advantageous for the synchronisation between the sender and the receiver and for the recovery of the transmission clock. This encoding method prohibits sending consecutive zeros or ones, which appear as constant DC signal. Since every bit of information is composed of two levels, a zero and one, the reference clock is at 20 MHz.

To identify the beginning of an Ethernet frame, a special pattern of bits, a preamble and a SFD (Start of Frame Delimiter), are sent prior to the actual data. It is composed of a series 64 bits of 1010101 ... 1011, of which the last two bits are 11. The last byte (SFD) is: 10101011. The preamble is 7 bytes of $0x55$ followed by one the SFD byte of $0xD5$. The first byte that is sent is $0x55$, whereas the byte $0xD5$ is sent last. The leftmost bytes are sent first, of which the rightmost bits (LSB) are sent first. This is why the first byte in the preamble : 10101010 is sent from right to left, as $0x55$, i.e., the first bit to be transmitted is 0. Generally, data are transferred from the FPGA to the Ethernet port through a physical interface (Phy). Taking into consideration the MII (Media Independent Interface)[6] standard, where the Phy interface communicates nibbles (4 bits)[7], the SFD 10101011 byte is sent as $0xD$ and $0x5$, since the lower nibble $0xD$ (1011) is sent first starting by 1 (rightmost bit).

The implementation of the behaviour described in this subsection on FPGA is through a HDL finite state machine (FSM), which is inspired by the open-source resources available at www.fpga4fun.com. The investigation of available open-source resources shortens the implementation time as explained by the design guideline 4 of section 7.5.

[6]The Gigabit MII (GMII) deals with 8 bits (reference clock at 125 MHz) at a time and the 10 Gbit MII (XGMII) standard deals with 32 bits at a time.

[7]The reference clocks are 2.5 MHz and 25 MHz for 10 Mbps and 100 Mbps, respectively. Two other reduced version of MII called RMII (Reduced MII) and SMII (Serial MII) for 2-bit and 1-bit bus widths, respectively.

7.4.3 Building Ethernet extension board prototypes

We have built several prototypes of the Ethernet extension board with free and available CAD tools to be easily connected to the numerous I/O pins of the D2SB FPGA board. By building several prototypes, we have had the opportunity to investigate the use of single-ended and differential signalling of the FPGA as well as to evaluate several schematic and layout CAD software, such as *geda schem*, *geda pcb*, *kicad*, *DipTrace* and *Eagle*. For example, the open-source *geda pcb and gschem* tools proved to be quite reliable. The schematic diagram of the final prototype, designed with *geda schem* as depicted on figure 7.3, shows the use of differential I/O pairs for the Tx and Rx pins and the single-ended crystal oscillator, for one of the expansion slots. The fabrication of the printed wiring boards (PWB) prototypes of the Ethernet extension interface (as can be seen on figure 7.3) on double-side boards involved household resources, such as laser-printer toner transfer, non toxic copper etching and tinning using electroplating for extended protection of the copper surfaces.

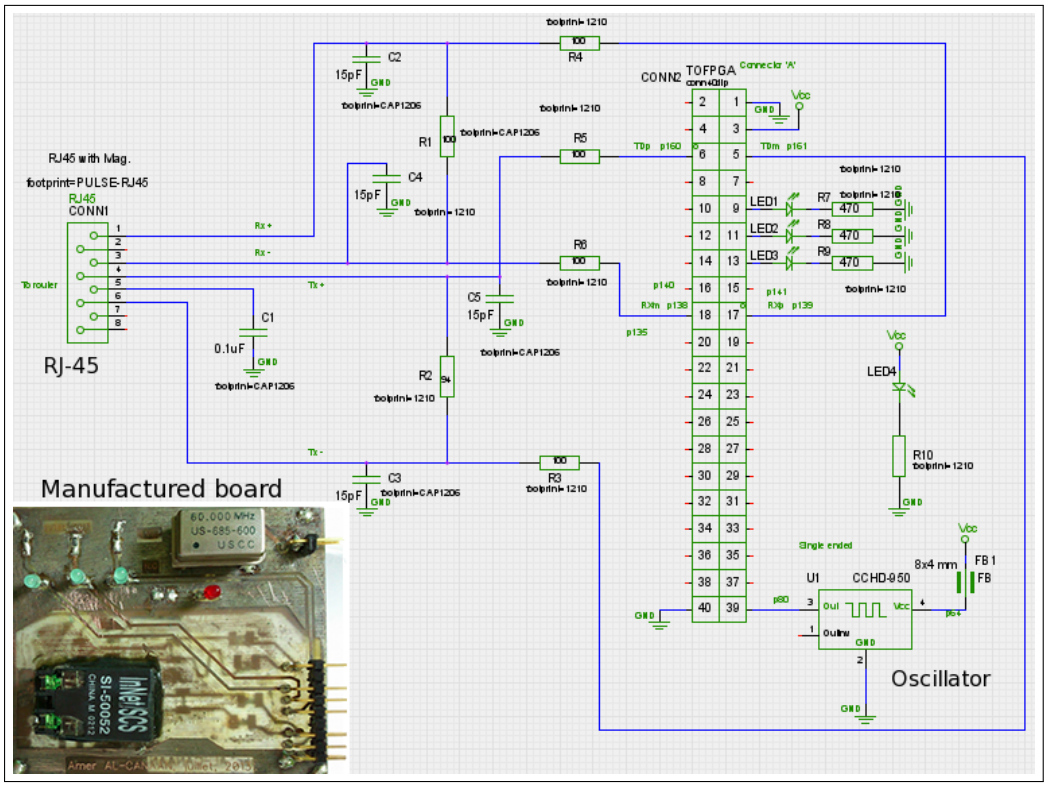


Figure 7.3 Schematic diagram for the developed Ethernet extension board and the manufactured board on the left.

The Ethernet extension board as shown on figure 7.3 consists of a RJ-45 connector (with an isolation transformer and integrated magnetics), a power-on LED indicator, three green LEDs connected to three FPGA I/O pins and a crystal oscillator chip.

We opted for an external crystal oscillator chip with 60 MHz frequency to provide an exact clock frequency and precise timing for our modules through the Delay Locked Loop (DLL) of the FPGA. It is obvious that the 60 MHz is an integer multiple of the 20 MHz, which makes it easy to divide. The characteristics of the 60 MHz oscillator are shown on figure 7.4, which shows that the second and third (120 MHz and 180 MHz) harmonics are well below the main frequency (60 MHz). This leads to the design guideline 5 of section 7.5.

7.4.4 Overview of software tools for FPGA

The design on FPGA necessitates specific software tools in order to synthesise HDL code into hardware programmed within the FPGA. The main software tools on FPGA include Xilinx[8] Webpack with *isim* simulator. Although, Xilinx is supported for both Windows and Linux, Mac OS X is not supported, unfortunately. We opted for Linux as our main development platform since Xilinx on Linux (Debian) proved to be more user-friendly and more effective.

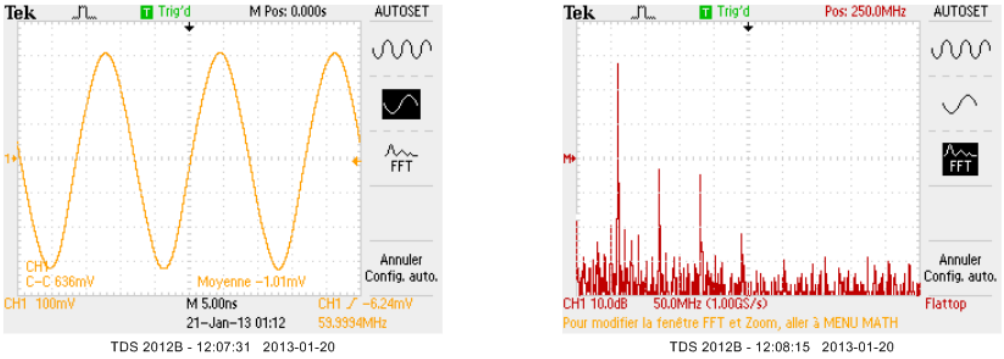
We used several packet-sniffing, packet-generation and testing tools including *Wireshark*, *packeth*. In addition, tools like *wget*, *curl* and *SocketTest*, which are used in section 7.6 for packet checking for TCP and HTTP protocols. This subsection leads to the design guideline 6 of section 7.5.

7.4.5 Test results and conclusion

An important step in the test phase, is to verify that the speed and area constraints are met through the adoption of coherent HDL coding style, as explained by the design guideline 7 of section 7.5.

In order to test our implementation of the UDP packet sender at 10 Mbps, an oscilloscope was used to capture the frames, where the preamble waveform appears as a distinctive

[8]We limit our discussion to Xilinx-related tools since we utilised only FPGA chips from Xilinx.



(a) Waveform. (b) Frequency spectrum.

Figure 7.4 The crystal oscillator characteristics for the Ethernet extension board.

periodic signal of 5 MHz, as depicted by the waveform on figure 7.5. The two red-horizontal lines on figure 7.5 are cursors and they are not part of the waveform.

Wireshark software is used to view the captured UDP and ARP packets, as shown on figure 7.6. In addition, some packets are selected and viewed on the LCD, for which we have utilised a FIFO memory and a dedicated state machine.

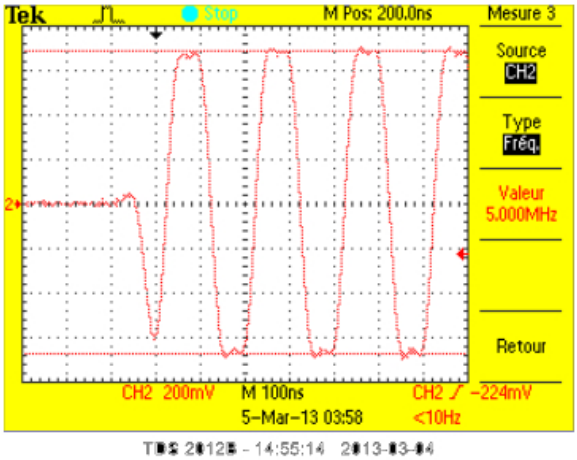


Figure 7.5 Waveform of a preamble of the 10 Mbps IP frame showing its distinctive 5 MHz frequency.

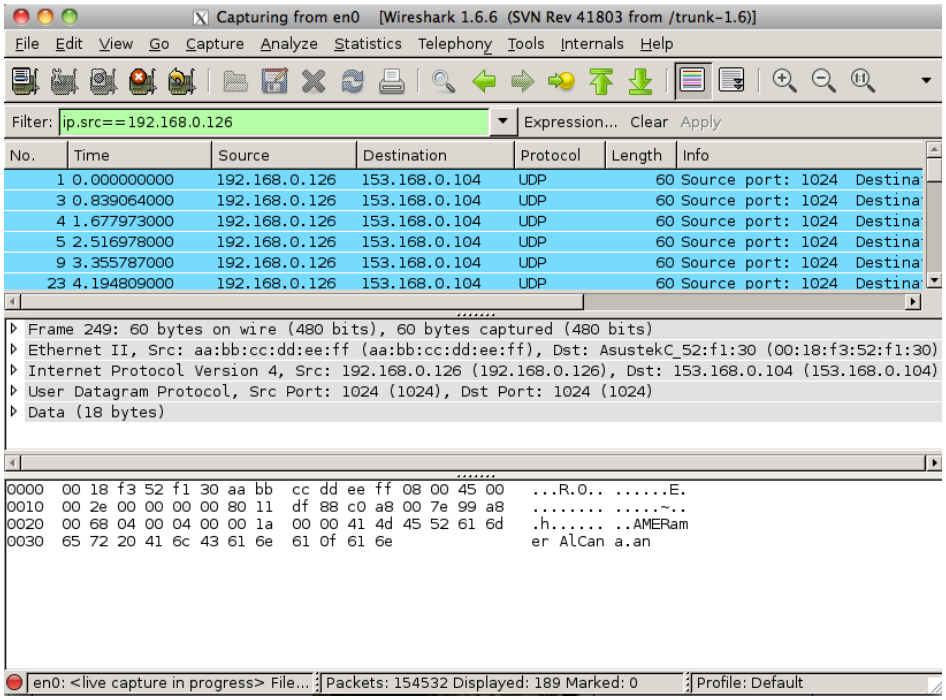


Figure 7.6 UDP packets as captured by *Wireshark*.

7.5 Design guidelines on FPGA

In section 7.4, we have presented an example of design on FPGA which has permitted to highlight 7 design guidelines. These guidelines are meant to help achieve new designs and implementations on FPGA and attend to clear the misconceptions about HDL, which may substantially delay designs. The seven design guidelines are presented below:

1. Proper verification of hardware requirements and availability for adequate hardware resources. This guideline is deduced from the first paragraph in section 7.4.1.

2. Careful study of the FPGA chip data sheet, in order to verify for the supported I/O standards, for instance, such as support for differential I/O signals. In our example this leads to simple interfacing with the Ethernet port, as shown in the third paragraph in section 7.4.1.

3. Careful study of the reference manuals and schematics of the FPGA kit (for example the D2SB or the Nexys3 board) as well as those of the peripherals utilised. Usually peripherals are configured in a more restricted way. For example, even though the FPGA provides enough number of differential pairs to be used, the FPGA kit may limit this number by wiring some of them as regular I/O to access other peripherals, such as LEDs and switches.

In another example, the Nexys3 board is pre-configured in MII only mode (as mentioned in subsection 7.6.1), while its Phy chip supports other modes. Though this limits the number of components (like resistors and oscillators, for example) and reduces cost, it has some influence on the choice of design of HDL modules. This guideline corresponds to the last two paragraphs in section 7.4.1.

4. Profit from available HDL modules for code-reuse is a time-saving strategy. Again, since FPGA design is hardware-dependent, the choice of hardware from the very start may greatly ease the implementation phase. The availability of open-source resources should be considered for inspiration or code adoption. It is proven that several commercial and successful HDL modules are based upon open-source modules. This guideline corresponds to the last paragraph in section 7.4.2.

5. Proper verification of the intended speed and clock precision. Though all FPGA kits provide a default oscillator chip, some of these are not crystal oscillators, which should be taken into consideration when high precision is needed. In addition to precision, the feasibility to generate the intended frequency may contribute to the choice of an external oscillator. This guideline corresponds to the last paragraph in subsection 7.4.3, where the 60 MHz can be easily divided to produce 20 MHz compared to the original 50 MHz.

6. The adoption of adequate FPGA software tools as well as development platforms helps alleviate the design and implementation delays and increase performance. This guideline corresponds to the section 7.4.4.

7. Since FPGA is viewed as programmable hardware, it is recommended to have some knowledge in hardware and logic design. Though the syntax of HDL has much in common with software and computer languages, HDL has its own particular nature and purpose. For this, the understanding of specific HDL performance tips [4–6] helps respect the area and speed constraints and produce efficient and bug-free application on FPGA. This guideline corresponds to the first paragraph in section 7.4.5.

7.6 Realisation of a high QoS MWS on FPGA

In this section, we provide details about the design and the implementation of our high QoS MWS. The first subsection provides details about the Ethernet module, while the

second subsection provides details about the Web service modules, as depicted on figure 7.7. We provide a complete design and realisation of a MWS on FPGA that concentrates mainly on the higher layers of the OSI model, whereas the previous design example in section 7.4 concentrates on the lower layers of the OSI. This section also concentrates on the service logic of the MWS as well as on several QoS parameters. The realisation of MWS on FPGA in this section depends upon the design guidelines presented in section 7.5, as well as on the Nexys3 platform, which has an integrated FPGA (Xilinx XC6SLX16 of the Spartan6 FPGA family) with several peripherals, as can be seen on figure 7.8. The availability of the RJ-45 connector and the Phy peripherals side by side on the Nexys3 board helps accelerate the implementation process. A significant advantage of this section (compared to the design example in section 7.4) is that it provides a design on a more compact FPGA platform that offers dedicated Ethernet and RAM peripherals, which help accelerate the design process and the integration of QoS modules.

7.6.1 Ethernet module

The basis for implementing MWS on FPGA is the Ethernet module, which allows a seamless communication between the FPGA and the Internet, as depicted on figure 7.7. Available commercial modules of communication between the Phy module and FPGA offer quick solutions. However, our choice of utilising open-source HDL code is justified by the flexibility and the unlimited access offered to the fine details of the code so that it can be modified and customised easily. We have investigated several open-source projects (cores), of which the Ethernet MAC 10/100 Mbps project (shown in Table 7.2) has been adopted for our MWS on the Nexys3 board. This module was selected for three reasons: 1) its compatible MII standard to the Phy available on the Nexys3 board, 2) its multiple features and 3) the number of contributors and the long period of development, since 2001. The first step of adapting this Ethernet module was to connect the Phy I/O signals to the FPGA through the user constraints file of the Nexys3 board.

Setting the operation mode of the Phy (i.e., 100 Mbps in full-duplex mode) on the Nexys3 board has been done through the addition of several I/O buffers. This has been achieved through a careful study of the data-sheet of the Phy chip on the Nexys3 Board (this corresponds to the design guideline 3 of section 7.5). We have developed a FSM to initialise the different configuration registers and provided input signals for the Wishbone in order to activate the transmission module. Besides, we have added two RAM modules

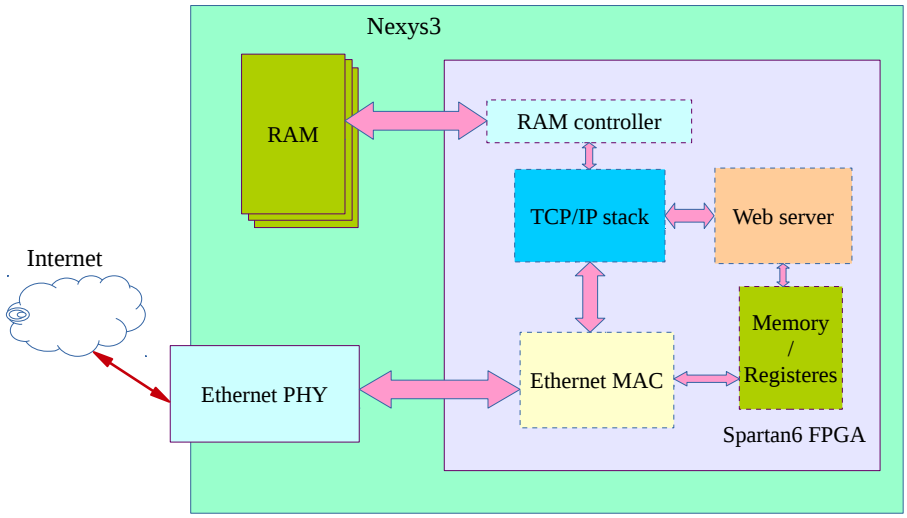


Figure 7.7 Architecture of the MWS on FPGA.

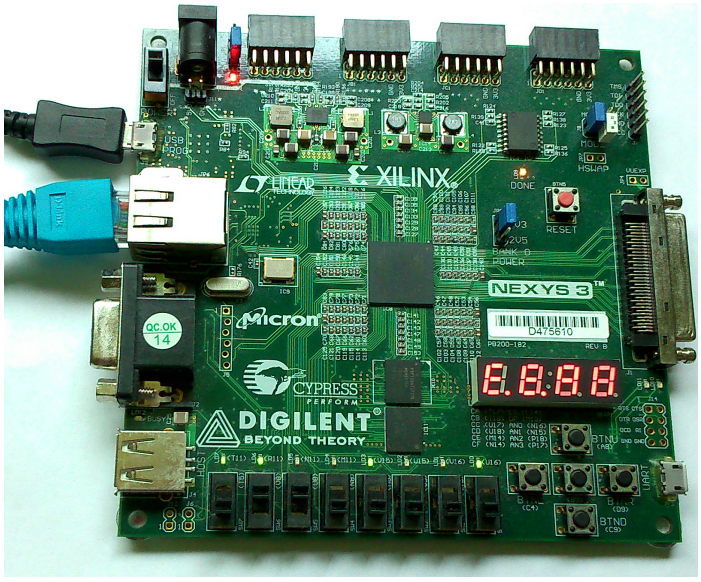


Figure 7.8 Overview of the Nexys3 board.

to save the received and transmit packets. For test purposes, we have added new modules in order to generate fixed-width UDP and ARP packets that are transmitted periodically every one second. This phase has been successfully achieved throughout the FSM for the Wishbone interface module that we have developed using StateCad. Although the Ethernet MAC 10/100 Mbps project offers a rich set of options and features, its documentation lacks some critical information about the operation of the Wishbone interface, which has caused extra delays for the design of the reception module.

Table 7.2 Comparison between different open-source Ethernet MAC cores[1].

Project Name	Speed (Mbps)	Phy type	Notes
Tri_mode Ethernet MAC	10[2]/100[2]/1000	GMII	Full duplex only for 1 Gbps
Ethernet MAC 10/100	10/100	MII	Complex Wishbone interface
Ethernet-FIFO converter	10/100	MII	Frame sizes suited for GSM
Minimac	10/100	MII	Simplified Wishbone interface
TCP/IP socket	10/100/1000	GMII/MII[3]	HTTP[4] example provided

[1] All tested projects are available at www.opencores.org or www.github.com.
[2] Half duplex.
[3] The MII support was added with close collaboration with the author.
[4] The HTTP stack is coded in C language and is converted to Verilog through python scripts.

Several adaptation steps have been necessary to meet our requirements. For example, we worked closely with the original author of the module in order to provide support for the MII Phy standard. We added more features to the HTTP stack including binary file transfer and carried out some optimisation to reduce FPGA resource consumption. A simple HTML interface of the MWS is shown on figure 7.9, which controls 4 LEDS, displays the state of the 8 switches and provides an estimation of the response time on the Nexys3 board.

7.6.2 Web service module

The Web service modules include a TCP/IP HDL stack, a Web server and a memory controller, as depicted on figure 7.7. In order to compensate for the delays encountered (in section 7.6.1) we opted for the adaptation of a second project core, namely, the TCP/IP

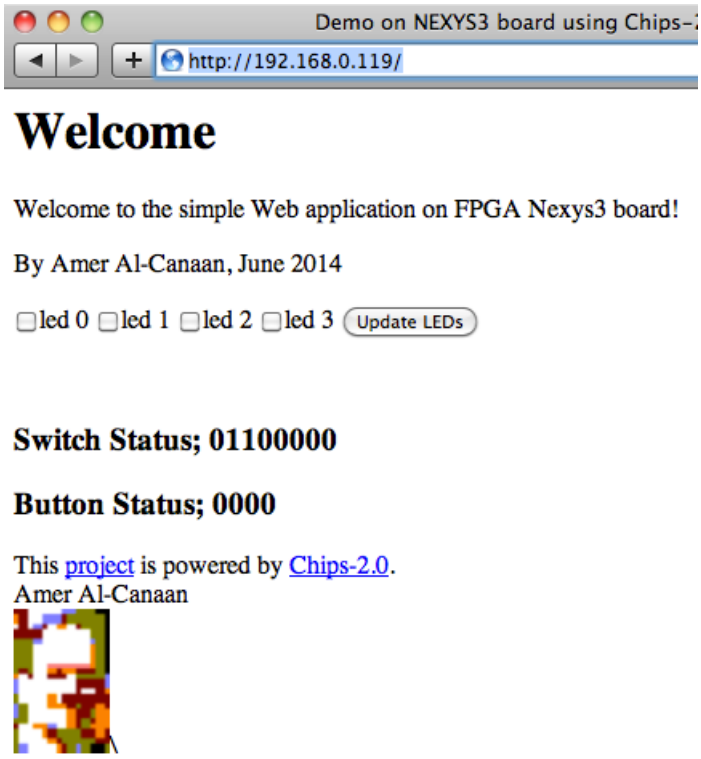


Figure 7.9 HTML Web page interface of the MWS.

socket, shown in table 7.2. The TCP/IP socket project is compelling since it offers a TCP/IP stack and a working example of a simple HTTP server, which offers a HTML page. The HTTP communications are provided through the high-level C programming language. The C code is converted through the Chips [7] tool, which is based upon python scripts to produce Verilog code (to be easily integrated within the FPGA synthesis software). We have modified the Web server in order to provide binary data instead of merely HTML. The binary data, images for instance, are stored inside the FPGA card.

Our created RESTful MWS provides multiple URLs for each resource that can be accessed through a Web browser or the command-line tools, such as *curl* and *wget*. The first URL interface as shown on figure 7.9 is through a Web page that allows to monitor the status of the first 4 LEDS and the 7 switches of the Nexys3 board. It also shows an embedded small image within HTML through the use of the inefficient base-64 encoding. Other URLs are utilised to retrieve binary files and images[9] through the utilisation of the 200 OK HTTP responses with binary payload. For large images, the chunked HTTP transfer is to be achieved.

[9]We limit the size of images up to 1 KB till the full implementation of the external memory controller on the Nexys3 board is completed. However we expect the FPGA to outperform the computer-version of the same MWS for large sized images.

7.7 Initial results and discussion

In this section, we present the initial results of our created RESTful image-retrieval MWS on FPGA at 100 Mbps (presented in section 7.6), since the TCP/IP stack is still in its development phase and is relatively new (released less than one year ago). The design effort is thus focused on the lower layers of the OSI model as well as on the higher layers. We highlight our initial results with minimal optimisation for the underlying TCP/IP stack.

Our test configuration platform is comprised of a wired LAN including two computers (Linux and Mac OS X) and a 100-Mbps switch, which enforces the maximum communication rate to be fixed at 100 Mbps. Each computer includes 1-Gbps Ethernet card and a dual-core microprocessor of at least 1GHz clock speed.

The summary output of ping commands as shown on Table 7.3, indicates an average value of $0.315\ ms$ for the round-trip time (RTT) for the PC$\rightarrow$FPGA, which is 35.65% below the PC$\rightarrow$PC reference value ($0.482\ ms$). The average round-trip times between the Linux or Mac and the router were the same. Although the multimedia Web service is running at 100 MHz clock frequency, the FPGA showed better performance (less RTT). The following lines show the raw output produced by the ping command with a 100 icmp packets for each possible configuration.

```
Mac-Linux--- 192.168.0.109 ping statistics ---
100 packets transmitted, 100 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.223/0.512/0.824/0.077 ms

Linux-Mac--- 192.168.0.106 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 98999ms
rtt min/avg/max/mdev = 0.223/0.452/0.617/0.091 ms

Mac-Switch--- 192.168.0.1 ping statistics ---
100 packets transmitted, 100 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.348/0.651/2.760/0.299 ms

Mac-Switch--- 192.168.0.1 ping statistics ---
100 packets transmitted, 100 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.348/0.651/2.760/0.299 ms

Linux-FPGA--- 192.168.0.119 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 98999ms
rtt min/avg/max/mdev = 0.166/0.279/0.500/0.101 ms

Mac-FPGA--- 192.168.0.119 ping statistics ---
100 packets transmitted, 100 packets received, 0.0% packet loss
```

```
round-trip min/avg/max/stddev = 0.384/0.531/0.676/0.056 ms

Linux-Routeur--- 192.168.0.1 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 98998ms
rtt min/avg/max/mdev = 0.204/0.313/0.475/0.073 ms
```

Table 7.3 Results of the RTT time in ms of the *ping* commands.

Configuration	**min**	**avg**	**max**	**stdev**	**Notes**
Mac → Linux	0.223	0.512	0.824	0.077	
Linux → Mac	0.223	0.452	0.617	0.091	
PC → PC	0.223	0.482	0.721	0.084	Linux or Mac
Mac → Switch	0.348	0.651	2.760	0.299	
Linux → Switch	0.204	0.313	0.475	0.073	
PC → Switch	0.276	**0.482**	1.618	0.186	Reference
Linux → FPGA	0.166	0.279	0.500	0.101	
Mac → FPGA	0.384	0.531	0.676	0.056	
PC → FPGA	0.275	**0.315**	0.588	0.079	35.65 % < Reference

In order to test and compare the MWS on FPGA, we created a similar MWS on the OS X computer using a HTTP server and the same set of images. The Wi-Fi interface on OS X was deactivated to ensure the same conditions with the client computer. During our tests, the behaviour of both MWS through the Linux client was identical. This means that with a low-power FPGA (powered through the USB port) at a main clock frequency of 100 MHz we easily obtain a similar performance as that obtained by a 2.4 GHz dual-core PC, consuming several hundreds of Watts.

When testing through Wi-Fi interfaces, the round-trip values become longer due to the bandwidth limit of the Wi-Fi interfaces. Since, the Nexys3 has a wired Ethernet port, we do not include the round-trip values measured through the Wi-Fi interface.

7.8 Conclusion

The parallelism, programmable in-memory design, affordable price, availability of design tools and low-power consumption make FPGA a highly-compelling alternative for computers in several domains, such as MWS, while providing excellent prototyping platforms for ASIC design and for multimedia Web services.
The widespread of parallel and pipeline solutions on FPGAs has opened the way for time-consuming and processing-intensive applications.

We have presented two design examples on FPGA, the first of which helps deduce a list of guidelines that are beneficial for designs of MWS on FPGA and is focused more on the lower layers of the OSI model. The second example represents a complete design and implementation of a RESTful MWS on FPGA, which concentrates more on the upper layers of the OSI model. Our strategy comprises pure HDL and open-source modules.

Though the comparison between FPGA and computer implementations, which are based upon the initial results obtained, reveal that for small sized images, both technologies show similar performance, whereas they differ enormously with respect to power consumption and clock speed. For example, the power consumed by the Nexys3 FPGA board in our examples (delivered by the USB2 interface and is usually below 10 Watts) is several folds less than the power consumed by a 13-inch MacBook Pro laptop computer (around 64 Watts). The frequency of the CPUs in our tests is 2400 MHz compared to 100 MHz for the MWS on FPGA, which shows an obvious advantage of designing on FPGAs. For larger payloads, the MWS on FPGA would outperform the computer counterparts by several folds, since the lager image sizes mean longer processing times, i.e., the MWS threads would be served more frequently by the CPU.

7.8.1 Future work

We plan to extend our approach to include more QoS parameters, other than those proposed and discussed in this paper, which include response time, download time and processing time. Our future work would concentrate on rapid compression, availability, throughput and security, for instance. After discussing the use of text and images, we would include more multimedia types, such as video and voice. Also, we would like to extend our MWS to include other modules to be integrated to the current architecture

including QoS management module and Web service workflow interactions module with minimum overhead cost.

The QoS management module performs negotiation, agreement and monitoring service of QoS parameters and of service interactions. The Web service interactions workflow module consists of prevention, detection and resolution of Web service interactions that may occur during operation. Interactions of services describe a non-desired state, in which services tend to compete on resources and influence negatively the operation and their intended behaviour. Obviously, this has a negative effect on QoS.

We plan to add more features to our MWS, such as enabling the transfer of huge files, by completing the implementation of the chunked-transfer encoding of the HTTP/1.1 protocol.

CHAPTER 8

Conclusion

Ce chapitre inclut les contributions (dans la section 8.1) et les travaux de recherches futurs proposés dans cette thèse (dans la section 8.2).

8.1 Contributions

La contribution de cette thèse peut se résumer comme suit :

1. Élaborer des méthodes de conception et de mise en œuvre de SWM sur des plate-formes logicielles hétérogènes, en particulier sur Windows XP, OS X et Solaris. Les méthodes de conception élaborées ont pour but d'offrir une portée étendue et des fonctionnalités variées et de nouveaux SWM tout en gardant une compatibilité maximale entre les plate-formes hétérogènes. Pour atteindre cet objectif, nous étudions les options possibles afin d'offrir des SWM riches en fonctionnalités nouvelles et mis en œuvre sur des plate-formes différentes. Pour ce faire, nous utilisons une interface de création de SWM indépendante de plate-formes. (Voir le chapitre 3).

2. Élaborer une liste de paramètres pertinents qui influencent la QdS selon les résultats obtenus lors de la mise en œuvre de SWM en utilisant le protocole SOAP (voir le chapitre 4) et le style REST (voir le chapitre 5). Ceci permet d'effectuer une comparaison précise décrivant le gain de performance lors de la mise en œuvre de SWM selon le style REST.

3. Élaborer une analyse de performance de SWM basés sur le protocole SOAP (voir le chapitre 4) et sur le style REST (voir le chapitre 5). L'analyse porte sur différents paramètres de QdS ainsi que sur différentes plate-formes et bandes passantes. Elle comporte l'utilisation de la compression binaire pour trois scénarios: local (à la machine même, appelé LOCAL), sur un réseau local (appelé LAN) et sur le réseau d'Internet (appelé WAN). Deux SWM ont été mis en œuvre avec une procédure spécifique permettant de cueillir les données de performance en temps réel.

L'analyse permet ainsi d'étudier le comportement de SWM réels comportant des images de tailles diversifiées soient entre 1×1 pixel jusqu'à 3822×4819 pixels. L'analyse propose d'intégrer des méthodes de compression de données binaires telles que DjVu dans les standards du Web de manière similaire à HTTP.

L'analyse est élargie pour couvrir d'autres paramètres de qualité de service liés à la vidéo et au son. (Voir le chapitre 5).

4. Analyser et étudier la corrélation entre la dimension d'une image et le taux de compression, en utilisant des outils et des libraires de manipulation d'image JPEG, JEPG2000 et DjVu. Les opérations effectuées sur les images incluent la compression, la modification de dimensions avec et sans déformations. À souligner que l'originalité de cette contribution est dû au fait qu'elle propose une nouvelle piste de recherche qui n'a pas été abordée auparavant par d'autres recherches connexes dans ce domaine. Les résultats de notre étude montrent qu'il est pertinent d'utiliser des images dont le ratio longueur/largeur respecte le ratio d'or (ou le chiffre d'or), $\frac{1+\sqrt{5}}{2}$, pour augmenter le taux de compression. Ces résultats sont confirmés pour des taux de compression élevés et modérés, c'est-à-dire supérieurs à 0.5. (Voir le chapitre 6).

5. Élaborer une méthode de conception et de mise en œuvre cohérente de SWM sur FPGA en utilisant des outils de conception HDL gratuits tels que Xilinx Webpack, des logiciels de simulation et du code *open-source*. Le choix de mise en œuvre sur FPGA est justifié par les caractéristiques intéressantes des cartes FPGA comme le parallélisme, la rapidité d'exécution, le coût abordable comparativement aux ASICs, car des solutions moins abordables et plus simples à mettre en œuvre (pas besoin de coder en HDL) existent, et la faible consommation d'énergie. Notre approche pour la mise en œuvre de SWM sur FPGA favorise l'utilisation du HDL (Verilog et VHDL) et l'utilisation de logiciels fournis gratuitement et du code à source ouverte, *open-source*. La mise en œuvre permet d'ajouter des modules plus complexes pour la gestion de la QdS et l'orchestration de SWM. (Voir le chapitre 7).

6. Mettre à jour ainsi qu'adapter du code HDL *open-source* (comme du code pour le module *TCP/IP socket*) et la documentation du module *Ethernet IP Core*, nécessaire pour la mise en œuvre de SWM sur FPGA dans le chapitre 7, pour la communication entre le FPGA et le port Ethernet sur la carte Nexys3. Ceci a pour effet de faciliter la communication entre FPGA et les périphériques. Ceci a aussi

pour effet de faciliter la mise en œuvre de SWM sur la carte Nexys3. (Voir les annexes A et B).

8.2 Travaux de recherches futurs

Les travaux futurs proposés suite aux travaux de cette thèse se résument comme suit :

- Étendre l'étude des paramètres qui influencent la QdS des SWM pour inclure des paramètres spécifiques aux réseaux mobiles, tels que la sécurité, la disponibilité et la perte de paquets.
- Proposer des SWM sécuritaires mis en œuvre sur FPGA à l'abri des attaques virales logicielles.
- Développer des modules de gestion de la QdS, de l'orchestration de SWM mis en œuvre sur FPGA ainsi qu'un module de compression d'images et de la vidéo. Ceci aura pour but de simplifier l'étape de conception et de mise en œuvre sur FPGA en utilisant exclusivement du code HDL (Verilog et VHDL).
- Étendre l'analyse de SWM mise en œuvre sur ordinateurs et sur FPGA pour inclure les SWM de diffusion vidéo.
- Proposer des améliorations sur l'architecture des SWM pour faciliter l'intégration et l'envoi de contenu binaire, comme les images et la vidéo qui ont des contraintes étroites de bande passante et de débit de transfert.
- Étendre les résultats de la méthode d'analyse du chapitre 6 pour étudier la corrélation entre les dimensions d'images et le nombre d'or lors de l'utilisation de diverses libraires de compression binaire (autres que les libraires JPEG, JPEG2000 et DjVu). Ceci aura pour but d'améliorer les taux de compressions.
- Proposer des algorithmes de compression d'images et de séquences vidéo adaptés sur FPGA pour offrir des SWM de qualité.
- Proposer des méthodes d'accélération de processus et d'algorithmes dans le domaine de traitement d'images mises en œuvre sur FPGA en profitant du parallélisme.

ANNEX A
La documentation du Ethernet IP CORE

Ethernet IP Core
Design Document

Author: Igor Mohor

Revised by Amer Al-Canaan

Rev. 0.5

March 23, 2014

Revision History

Rev.	Date	Author	Description
0.1	09/09/02	Igor Mohor	First draft
0.2	22/10/02	Igor Mohor	Description of Core Modules added (figure), Some test description added.
0.3	29/10/02	Igor Mohor	Some figures added.
0.4	29/10/02	Igor Mohor	Description of test cases added.
0.5	04/04/014	Amer Al-Canaan	General revision

A.1 Ethernet IP Core Introduction

The Ethernet IP Core is a MAC (Media Access Controller). It connects to the Ethernet PHY chip on one side and to the WISHBONE SoC bus on the other. The core has been designed to offer as much flexibility as possible to all kinds of applications.

The section A.4 describes file hierarchy, description of modules, core design considerations and constants regarding the Ethernet IP Core.

The section A.7 describes test bench file hierarchy, description of modules, test bench design considerations, description of test cases and constants regarding the test bench.

A.2 Ethernet IP Core Features

The following lists the main features of the Ethernet IP core.

- Performing MAC layer functions of IEEE 802.3 and Ethernet
- Automatic 32-bit CRC generation and checking
- Delayed CRC generation
- Preamble generation and removal
- Automatically pad short frames on transmit
- Detection of too long or too short packets (length limits)
- Possible transmission of packets that are bigger than standard packets.
- Full duplex support
- 10 and 100 Mbps bit rates supported
- Automatic packet abortion on Excessive deferral limit, too small inter-packet gap, when enabled
- Flow control and automatic generation of control frames in full duplex mode (IEEE 802.3x)
- Collision detection and auto re-transmission on collisions in half duplex mode (CSMA/CD protocol)
- Complete status for TX/RX packets
- IEEE 802.3 Media Independent Interface (MII)
- WISHBONE SoC Interconnection Rev. B2 and B3 compliant interface
- Internal RAM for holding 128 TX/RX buffer descriptors
- Interrupt generation an all events

A.3 Ethernet IP Core Directory Structure

The following picture shows the structure of directories of the Ethernet IP core.

There are two major parts of the Verilog code in the **ethernet** directory. First one is the code for the Ethernet MAC IP core. The Verilog files are in the **ethernet\rtl\verilog** sub-directory. The second one is the code for the Ethernet MAC Test-bench. These files are used together with files for the Ethernet MAC. There are also some exceptions,

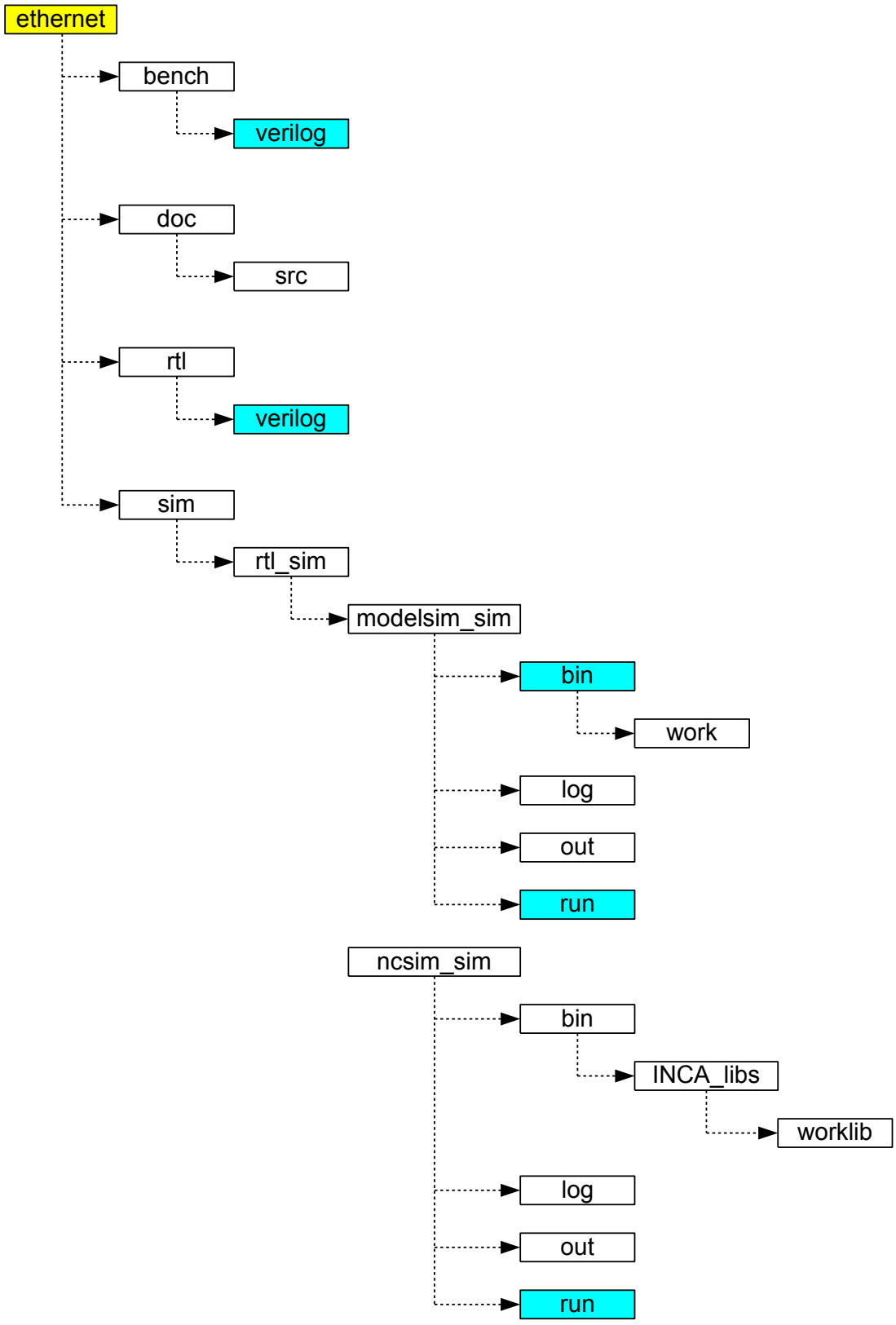


Figure A.1 Ethernet IP Core Core Directory Structure

but those will be mentioned later. The Verilog files are in the **ethernet\bench\verilog sub-directory**.

The documentation is in the sub-directory **ethernet\doc**. Documentation consists of Ethernet IP Core Data Sheet, Ethernet IP Core Specification and Ethernet IP Core Design document.

ethernet\sim sub-directory is used for running simulation – testbench. The **rtl_sim** sub-directory is used for RTL (functional) simulation of the core. There are two sets of scripts for running the simulation. First set is used for running the test-bench using NCSim simulator. Second set is used for running the test-bench using ModelSIM simulator. Both are using the similar directory structure:

- **bin** – includes various scripts needed for running Ncsim simulator
- **run** – the directory from which the simulation is run. It provides a script for starting the simulation and a script for cleaning all the results produced by previous simulation runs
- **log** – Ncvlog, Ncelab and Ncsim log files are stored here for review.
- **out** – simulation output directory – simulation stores all the results into this directory (dump files for viewing with Signalscan, testbench text output etc.)

Generated files from synthesis tools, like gate level Verilog and log files, are stored in the **ethernet\syn** sub-directory and its sub-directories.

A.4 Ethernet MAC IP Core

A.4.1 Overview

The Ethernet MAC IP Core consists of seven main units: WISHBONE interface, transmit module, receive module, control module, MII module, status module and register module. Many of these modules have sub-modules. Module and sub-module operations are described later in this section.

WISHBONE Interface

Consists of both master and slave interfaces and connects the core to the WISHBONE bus. Master interface is used for storing the received data frames to the memory and loading the data that needs to be sent from the memory to the Ethernet core. Interface is WISHBONE Revision B.2 and B.3 compatible (select-able with a define ETH_WISHBONE_B3 in the eth_defines.v file).

A.4.2 Transmit Module

Performs all transmitting related operations (preamble generation, padding, CRC, etc.).

A.4.3 Receive Module

Performs all reception related operations (preamble removal, CRC check, etc).

A.4.4 Control Module

Performs all flow control related operations when Ethernet is used in full duplex mode.

A.4.5 MII Module (Media Independent Module)

Provides a Media independent interface to the external Ethernet PHY chip.

A.4.6 Status Module

Records different statuses that are written to the related buffer descriptors or used in some other modules.

A.4.7 Register Module

Registers that are used for Ethernet MAC operation are in this module.

A.5 Core File Hierarchy

The hierarchy of modules in the Ethernet core is shown here with file tree. Each file implements one module in a hierarchy. RTL source files of the Ethernet core are in the **ethernet\rtl\verilog** sub-directory.

ethernet

. sim

. . rtl_sim

. . . src

. . . run

. rtl

. . verilog

. . . eth_top.v

. . . eth_crc.v

. . . eth_cop.v

. . . eth_miim.v

. . . eth_defines.v

. . . timescale.v

. . . eth_random.v

. . . eth_fifo.v

. . . eth_wishbone.v

. . . eth_maccontrol.v

. . . eth_rxaddrcheck.v

. . . eth_txstatem.v

. . . eth_transmitcontrol.v

. . . eth_txethmac.v

. . . generic_spram.v

. . . eth_rxcounters.v

. . . eth_rxstatem.v

. . . eth_outputcontrol.v

. . . eth_register.v

. . . eth_receivecontrol.v

. . . eth_registers.v

. . . eth_shiftreg.v

. . . eth_txcounters.v

. . . eth_clockgen.v

. . . eth_rxethmac.v

. . . eth_macstatus.v

. doc

. . eth_speci.pdf

. . eth_design_document.pdf

. . Ethernet Datasheet (prl.).pdf

. . src

. . . eth_speci.doc

. . . eth_design_document.doc

. . . Ethernet Datasheet (prl.).doc

. bench

. . verilog

. . . tb_ethernet.v

. . . tb_eth_defines.v

. . . tb_cop.v

. . . eth_host.v

. . . eth_memory.v

A.6 Description of Core Modules

The module **eth_top.v** consists of the following sub modules:
eth_miim.v, **eth_registers.v**, **eth_maccontrol.v**, **eth_txethmac.v**, **eth_rxethmac.v**, **eth_wishbone.v**, **eth_macstatus.v** and some logic for synchronising, multiplexing and registering outputs.

A.6.1 Description of the MII module (eth_miim.v)

The MII module (Media Independent Interface) is an interface to the external Ethernet PHY chip. It is used for setting PHY's configuration registers and reading status from it. The interface consists of only two signals: clock (**MDC**) and bi-directional data signal (**MDIO**). Bi-directional MDIO signal needs to be combined from input signal **Mdi**, output signal **Mdo**, and enable signal **MdoEn** in additional module. This is done because the same Ethernet core will be implemented in both ASIC and FPGA.

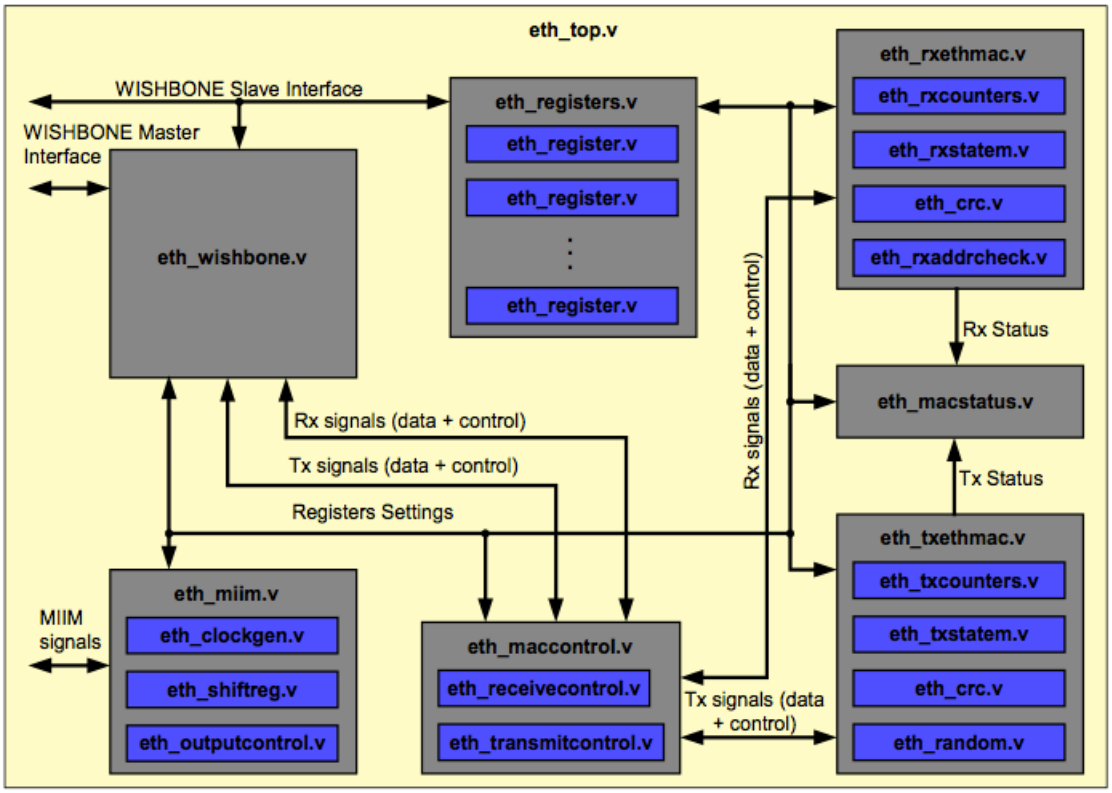


Figure A.2 Core Modules

The MII module is the top module for the MII and consists of several sub modules (**eth_clockgen.v**, **eth_shiftreg.v**, **eth_outputcontrol.v)** and additional logic. This logic is used for generating number of signals:

- Synchronised request for write (**WriteDataOp**), read (**ReadStatusOp**) and scan (**ScanStatusOp**) operations.
- Signal for updating the MIIRX_DATA register (**UpdateMIIRX_DATAReg**)
- Counter (**BitCounter**) is the primary counter for the MII Interface (many operations depend on it.
- Byte select signals used when data is shifted out (**ByteSelect [3:0]**).
- Signals used for latching the input data (**LatchByte [1:0]**).

When there is a need to read or write the data from the PHY chip, several operations need to be performed:

- MIIMODER register needs to be set:
  - ☐ Clock divider needs to be set to provide clock signal **Mdc** of the appropriate frequency (read PHY documentation to obtain the value of the Mdc frequency)

- ☐ Preamble generation might be disabled (if PHY supports transmissions without the preamble). By default 32-bit preamble is transmitted.
- ☐ MII Module might be reset prior to its usage.

- PHY Address (several PHY chips might be connected to the MII interface) and address of the register within the selected PHY chip need to be set in the MIIADDRESS register.
- If there is a need to write data to the selected register, data needs to be written to the MIITX_DATA register.
- Writing appropriate value to the MIICOMMAND register starts requested operation.
- If “Read status” or ”Scan status” operation were requested than the value that was received from the PHY can be read from the MIIRX_DATA register.

MIISTATUS register reflects the status of the MII module. The **LinkFail** status is cleared only after the read to the PHY’s status register (address 0x1) returns status that is OK.

Description of the eth_outputcontrol module

This module performs two tasks:

- Generates MII serial output signal (**Mdo**)
- Generates enable signal (**MdoEn**) for the **Mdo**.

Since the MII serial data signal is a bi-directional signal, these two signals need to be combined together with the MII serial input signal (**Mdi**) in additional module that is not part of the Ethernet MAC IP Core.

The eth_outputcontrol module also generates the MII preamble. When MII preamble is enabled (bit 8 in the MIIMODER register set to 0), 32-bit preamble is transmitted prior to the data.

Description of the eth_clockgen module

The eth_clockgen module is used for:

- Generating MII clock signal (**Mdc**). This is output clock signal used for clocking the MII interface of the Ethernet PHY chip. You should read the specification for the used PHY chip to properly set the Mdc frequency (usually frequencies up to 10 MHz can be used)
- Generating **MdcEn** signal. This signal is an enable signal. All flip-flops used in the MII are clocked with the high frequency clock **Clk**. The reduced frequency (equal to Mdc) is obtained by using the MdcEn signal.

Mdc is obtained by dividing the **Clk** signal with the value that is written in the MIIMODER register (any value within range [1:255]).

Description of the eth_shiftreg module

The eth_shiftreg module is used for:

- Serialise the data that goes towards Ethernet PHY chip (Mdo)
- Parallelize input data that comes from Ethernet PHY chip (Mdi) and temporally store it to the **Prsd** register. This value is then stored to the MIIRX_DATA register.
- Generating LinkFail signal (bit 0 of the MIISTATUS register reflects its value).

A.6.2 Description of the Receive module (eth_rxethmac.v)

The Receive module is in charge for receiving data. External PHY chip receives serial data from the physical layer (cable), assembles it to nibbles and sends to the receive module (**MRxD [3:0]**) together with the "data valid" marker (**MRxDV**). The receive module then assembles this data nibbles to data bytes, and sends them to the WISHBONE interface module together with few signals that mark start and end of the data. Receive module also removes the preamble and the CRC.

The Receive module consists of four sub modules:

- **eth_crc** – Cyclic Redundancy Check (CRC) module
- **eth_rxaddrcheck** – Address recognition module
- **eth_rxcounters** – Various counters needed for packet reception
- **eth_rxstatem** – State machine for Receive module

Besides the above sub modules, eth_rxethmac module also consists of logic that is used for:

- Generating **CrcHash** value and **CrcHashGood**marker that are used in address recognition system.
- Latching the data that is received from the PHY chip (**RxData**).
- Generating **Broadcast** and **Multicast** marker (when packets with broadcast or multicast destination address are received).
- Generating **RxValid**, **RxStartFrm**, **RxEndFrm**signals that are marking valid data.

Receiver can operate in various modes. For that reason number of registers need to be configured prior to Receiver's use.

Signals related to the receiver operation are:

- **HugEn** – Reception of big packets is enabled (packets, bigger than the standard Ethernet packets). When HugEn is disabled, packets that smaller or equal to MaxFL and bigger or equal to MinFL are received. (MaxFL and MinFL are set in the PACKETLEN register).
- **DlyCrcEn** – Delayed CRC (Cyclic Redundancy Check) is enabled. CRC checking starts 4 bytes after the data becomes valid. This option is useful when additional data is added to the data frame.
- **r_IFG** – Minimum Inter Frame Gap Enable. When this signal is set to zero, minimum inter frame gap is required between two packets. After this time receiver starts with reception again. When r_IFG is set to 1, no inter packet gap is needed. All frames are received regardless to the IFG.
- **r_Pro, r_Bro, r_Iam** and registers **MAC, HASH0** and **HASH1** are used for address recognition.

Description of the CRC (Cyclic Redundancy Check) module (eth_crc.v)

This module is used for validating the correctness of the incoming packet by checking the CRC value of the packet. CRC module is also used for the CRC generation for the TX module.

To better understand the CRC checking, here is a brief description how CRC is sent and checked.

Before a transmitter sends the data, it appends the CRC (this CRC is calculated from the data) to it. This means that the packet is now bigger for 4 bytes. Receiver receives this data (that **also includes the CRC of the data**) and calculates a new CRC value from it (received CRC is also used for the CRC calculation). If the new CRC differs from the "CRC Magic Number" (0xc704dd7b), then received data differs from the sent data and **CrcError**signal is set.

Description of the address recognition module (eth_rxaddrcheck.v)

The address recognition module decides whether the packet will be received or not. Ethernet IP core starts receiving all packets regardless to their destination address. Destination address is then checked in the eth_rxaddrcheck sub module. Frame reception depends on few conditions:

- If **r_Pro** bit is set in the MODER register (Promiscuous mode), then all frames are received regardless to their destination address. If r_Pro bit is cleared then destination address is checked.

- If **r_Bro** bit is set in the MODER register then all frames containing broadcast addresses are rejected (r_Pro must be cleared).
- **MAC** – MAC address of the used Ethernet MAC IP Core. This is individual address of the used Ethernet core. When r_Pro bit is cleared then every destination address is compared to the MAC address. Frame is accepted only when the two addresses match.
- When **r_Iam** signal is set then besides checking the MAC address, hash table algorithm is used. The Ethernet controller maps any 48-bit address into one of 64 bits. If that bit is set in the HASH registers (**HASH0** and **r_HASH1** are making one 64-bit hash register), then frame is accepted.

As said before, packet reception always starts regardless of the destination address of the incoming packet. As soon as the destination address is received, it is checked if it matches with any of the above-mentioned conditions. If the match doesn't occur than the reception of the whole packet is aborted (signal **RxAbort** is set to 1). The packet is not written to the memory and receive buffer is flushed.

Description of the rxcounters module (eth_rxcounters.v)

The module consists of three counters, which are:

- **ByteCnt** – generally used counter in the receive module.
- **IFGCounter** – used for counting the IFG (inter frame gap)
- **DlyCrcCnt** – counter, used when delayed CRC operation is enabled.

Besides that a number of comparators are in this module, used for various purposes.

Description of the rxstatem module (eth_rxstatem.v)

There is just one state machine used in the receive module of the Ethernet IP core. This module is placed in the eth_rxstatem sub-module.

The state machine has six different states:

- Idle state
- Drop state
- Preamble state
- SFD (standard frame delimiter) state
- Data 0 state
- Data 1 state

State machine (SM) goes to the drop state (**StateDrop**) after the reset and immediately after that to the idle state (**StateIdle**) because **MRxDV** is set to 0. As soon as there is a valid data available on the PHY's data lines (**MRxD**), PHY informs receiver about that by setting the **MRxDV**signal to one.

Normally receiver expects preamble at the beginning of each packet. Standard preamble is 7 byte long (0xee). After that a one-byte SFD (start frame delimiter) is expected (0xde). If we put this together, then sample 0xdeeeeeee is expected (LSB received first).

Because the Ethernet IP core can also accept packets that don't have a standard 7-byte preamble but only the SFD, receiver's SM waits for the first 0x5 nibble (it is not important whether this nibble is part of the preamble or of the SFD). If the received character differs from the expected nibble, then the SM goes to the preamblestate (**StatePreamble**) and remains there until the correct nibble (0x5) is received. Once the 0x5 nibble is received, SM goes to the SFD state (**StateSFD**) where it waits for the 0xd nibble.

From here two things, depending on the value of the **IFGCounterEq24** signal, may occur (next paragraph describes IFGCounterEq24 signal).

If IFGCounterEq24 is set then:

- SM goes to the data0 state (**StateData0**) where lower data nibble is received and then to the data1 state (**StateData1**) where higher data nibble is received. SM goes back to the data0 state. SM continues going from data state 0 to data state 1 and vice versa until whole data packet is received and end of packet is detected (PHY clears the **MRxDV**signal). Once the data valid signal is cleared, SM goes to the idle state (**StateIdle**) and everything starts again.

else (IFGCounterEq24 is cleared):

- SM goes to the drop state (**StateDrop**) and remains there until the end of valid data is reported (PHY clears the **MRxDV**signal). After that SM goes to the idle state (**StateIdle**) and everything starts again.

Signal **IFGCounterEq24** is used for detecting the proper gap between two consecutive received frames (Inter Frame Gap). By the standard this gap must be at least 960 ns for 100 Mbps mode or 9600ns for 10 Mbps mode. If the gap is appropriate (equal or greater than requested), then **IFGCounterEq24** is set to 1. Signal **IFGCounterEq24** is also set to 1, when IFG bit in the MODER register is set (minimum inter frame gap is not checked). If the IFG gap between two frames is too small, frame won't be accepted but dropped.

A.6.3 Description of the Transmit module (eth_txethmac.v)

The Transmit module (TX) is in charge for transmitting data. TX module gets data that needs to be transmitted from WISHBONE interface (WBI) module in the byte form. Besides that it also receives signals that mark start of the data frame (**TxStartFrm**) and

end of the data frame (**TxEndFrm**). As soon as the TX module needs next data byte, it sets the **TxUsedData** and WBI module provides the next byte.

TX module sets number of signals to inform WBI module on one side and Ethernet PHY chip on the other about the operation status (done, retry, abort, error, etc.).

The Transmit module consists of four sub modules:

- **eth_crc** – Cyclic Redundancy Check (CRC) module generates 32-bit CRC that is appended to the data field.
- **eth_random** – Generates random delay that is needed when back off is performed (after the collision)
- **eth_txcounters** – Various counters needed for packet transmission
- **eth_txstatem** – State machine for TX module

Signals, connected to the Ethernet PHY chip are:

- Data nibble **MTxD**. This is the data that will be sent on the Ethernet by the PHY.
- Transmit enable **MTxEn** tells PHY that data **MTxD** is valid and transmission should start.
- Transmit error **MTxErr** tells PHY that an error happened during the transmission.

Signals, connected to the upper layer module (WBI module) are:

- Transmit packet done **TxDone**(see next paragraph)
- Transmit packet retry **TxRetry**(see next paragraph)
- Transmit packet abort **TxAbort**(see next paragraph)
- **TxUsedData**

Every transmission ends in one of the following ways:

- Transmission is successfully finished. Signal **TxDone** is set.
- Transmission needs to be repeated. Signal **TxRetry** is set. This happens when a normal collision occurs (in half-duplex mode).
- Transmission is aborted. Signal TxAbort is set. This happens in the following situations:
  - □ Packet is too big (bigger than the max. packet (See MAXFL field of the PACKETLEN register)).
  - □ Underrun occurs (WBI module can not provide data on time).
  - □ Excessive deferral occurs (TX state machine remains in the defer state for too long).

- ☐ Late collision occurs (late collision is every collision that happens later than COLLVALID bytes after the preamble (See COLLCONF register)).
- ☐ Maximum number of collisions happens (See MAXRET field of the COLLCONF register).

Besides all previously mentioned signals, TX module provides other signals:

- **WillTransmit** notifies the receiver that transmitter will start transmitting. Receiver stops receiving until WillTransmit is cleared.
- Generating the collision reset signal (“collision detected” asynchronously comes from the PHY chip and is synchronized to the TX clock signal). **ResetCollision** signal is used to reset synchronising flip-flop.
- Collision window **ColWindow** marks a window within every collision is treated as a valid (regular) collision. After a collision packet is re-transmitted. Every collision that occurs after that is a late collision (packets with late collision are aborted).
- Retry counter **RetryCnt**.
- **Data_Crc, Enable_Crc** and **Initialize_Crc** that are used for CRC generation.

Description of the CRC (Cyclic Redundancy Check) module (eth_crc.v)

This module is used for CRC calculation. The calculated CRC is appended to the data frame. This module is also used in the RX module for CRC checking.

Description of the random module (eth_random.v)

When a collision occurs, TX module first sends a “jam” pattern (0x99999999) and then stops transmitting. Before a re-transmission starts, TX performs a back-off. TX waits before it starts transmitting for some amount of time. The amount of time is “semi” random and is calculated in the eth_random module. Binary Exponential algorithm is used for that purpose. Back-off time is random within predefined limits. This limits increase with the number of collisions.

Description of the TX counters module (eth_txcounters.v)

There are three counters in the eth_txcounters module. These counters are only used in the TX modules.

The **DlyCrcCnt** counter is used when a delayed CRC generation is needed to count

The nibble counter **NibCnt** count nibbles while **ByteCnt** counts bytes. Which one of the counters is used depends off the needed resolution.

Description of the TX state machine module (eth_txstatem.v)

The TX module has one general state machine that is in the eth_txstatem module. This state machine has eleven states:

- StateIdle
- StatePreamble
- StateData0
- StateData1
- StatePAD
- StateFCS
- StateIPG
- StateJam
- StateJam_q
- StateBackOff
- StateDefer

After the reset defer state (**StateDefer**) is activated. After that the state machine goes to the “Inter Packet Gap” state (**StateIPG**) and then to the idle state (**StateIdle**). Why this is so, is not important at the moment.

Let’s start with the description after the state machine comes to the idle state. This is the most often used state. When transmitter has nothing to do, it waits in the idle mode for the transmission request. Wishbone Interface (WBI) requests the transmission by setting the **TxStartFrm** signal to 1 for two clock cycles (together with the first byte of the data that needs to be sent). This forces the state machine (SM) to go to the preamble state (**StatePreamble**). In the preamble state **MTxEn** signal is set to 1, informing the Ethernet PHY chip that transmission will start. Together with the MTxEn signal, data signal **MTxD** is set to the preamble value 0x5. After the preamble is sent (0x5555555), SFD is sent (Start Frame Delimiter (0xd)). After that SM goes to the data0 state (**StateData0**) and signal **TxUsedData** is set to inform the WBI to provide next data byte. LSB nibble of the data byte is sent and then SM goes to the data1 state (**StateData1**), where the MSB nibble of the data byte is sent. SM continues to switch between the data0 and data1 states until the end of the packet. When there is just one byte left to be send, WBI sets the signal **TxEndFrm** that marks the last byte of the data that needs to be sent.

From here, there are several possibilities:

- If the **data length**is **greater or equal** to the minimum frame length (value written in the **MINFL** field of the PACKETLEN register) and **CRC** is **enabled** (bit CRCEN in the MODER register is set to 1 **or** bit CRC of the transmit descriptor is set to 1)

then SM goes to the **StateFCS** state where the 32-bit CRC value, calculated from the data, is appended. Then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.

- If the **data length**is **greater or equal** to the minimum frame length (value written in the **MINFL** field of the PACKETLEN register) and **CRC** is **disabled** (bit CRCEN in the MODER register is set to 0 **and** bit CRC of the transmit descriptor is set to 0) then SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.
- If the **data length**is **smaller** than the minimum frame length (value written in the **MINFL** field of the PACKETLEN register) and **padding**is **enabled**(bit PAD in the MODER register is set to 1 **or** bit PAD of the transmit descriptor is set to 1), then the SM goes to the pad state (**StatePAD**) where data is padded with zeros until the minimum frame length is achieved. Then the SM goes to the **StateFCS** state where the 32-bit CRC value, calculated from the data, is appended. Then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.
- If the **data length**is **smaller** than the minimum frame length (value written in the **MINFL** field of the PACKETLEN register), **padding**is **disabled**(bit PAD in the MODER register is set to01 **and** bit PAD of the transmit descriptor is set to 0) and **CRC** is **enabled** (bit CRCEN in the MODER register is set to 1 **or** bit CRC of the transmit descriptor is set to 1) then the SM goes to the **StateFCS** state where the 32-bit CRC value, calculated from the data, is appended. Then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.
- If the **data length**is **smaller** than the minimum frame length (value written in the **MINFL** field of the PACKETLEN register), **padding**is **disabled**(bit PAD in the MODER register is set to01 **and** bit PAD of the transmit descriptor is set to 0) and **CRC** is **disabled** (bit CRCEN in the MODER register is set to 0 **and** bit CRC of the transmit descriptor is set to 0) then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.

A.6.4 Description of the Control module (eth_maccontrol.v)

The Control module is in charge for data flow control, when Ethernet IP Core is in the 100 Mbps full duplex operating mode.

Control module consists of multiplexing logic and two sub modules:

- **eth_transmitcontrol**

- **eth_receivecontrol**

Flow control is done by sending and receiving **pause control frames**.

When the device that is connected to the WISHBONE interface of Ethernet IP Core (usually a processor) cannot process all those packets that it has received (and is still receiving), it requests a **pause** from the other station that is sending packets. The pause is requested by sending a pause control frame to the other station (see Ethernet IP Core Specification for details about the control frame). As soon as the other station receives pause request, it stops transmitting. The transmission is restarted after the requested pause time passes or pause request is switched off. The transmit flow control is done in the **eth_transmitcontrol** module. See description of the eth_transmitcontrol module for more details.

When the Ethernet IP Core receives a pause request, it stops transmitting for the requested time. This is done in the **eth_receivecontrol** module. See description of the eth_receivecontrol module for more details.

Multiplexing logicis used for multiplexing data and control signal used in normal transmission with data and control signals used for control frame transmission (see signals **TxUsedDataOut, TxAbortOut, TxDoneOut, TxEndFrmOut, TxStartFrmOut**).

When control frames are sent, padding and CRC generation is automatically switched on (see **PadOut** and **CrcEnOut** signals).

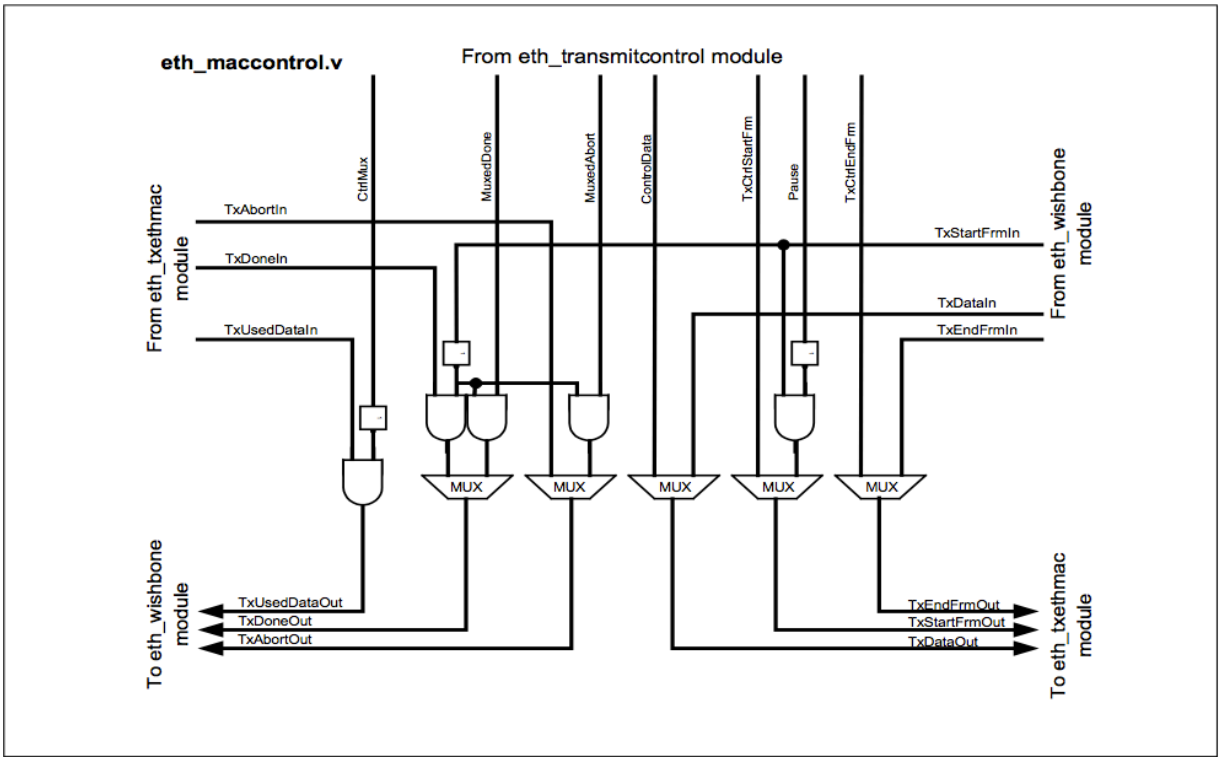


Figure A.3 Multiplexing Data and Control Signals in Control Module

Description of the CRC (Cyclic Redundancy Check) module (eth_crc.v)

This module is used for CRC calculation. The calculated CRC is appended sto the

A.6.5 Description of the Status module (eth_macstatus.v)

The Status module is in charge for monitoring the Ethernet MAC operations. Module monitors several conditions and after every completed operation (received or sent frame), it writes a status to the related buffer descriptor. Not all statuses are written to the buffer descriptors. See the following sections for more details.

Statuses for received frames are normally latched at the end of the reception stage (when signal TakeSample goes to 1). Soon after that statuses are reset (when signal LoadRxStatus goes to 1).

Rx Error (LatchedMRxErr)

This error notifies that the PHY detected an error while receiving a frame. In this case frame reception is aborted and no error reported. When invalid symbol is accepted, frame is still received and invalid symbol error reported in the Rx BD.

Rx CRC Error (LatchedCrcError)

This error notifies that a frame with invalid CRC was received. Such frame is normally received except that the CRC error status is set in the related Rx BD. If received frame is a control frame (pause frame), then pause timer value is not set.

Rx Invalid Symbol (InvalidSymbol)

This error notifies that a frame with invalid symbol was received. Invalid symbol is reported by the PHY when it is operating in the 100 Mbps mode (PHY sets data lines to 0xe when symbol error is detected).

Rx Late Collision (RxLateCollision)

When a late collision occurs, frame is normally received and late collision reported in the Rx BD. Late collision reflects the abnormal operation on the Ethernet (should never happen). See COLLCONF register in the Ethernet IP Core Specification for more details about the late collision.

Rx Short Frame (ShortFrame)

Short frames are normally (by default) aborted. This means that their appearance is not recorded anywhere. However if their reception is enabled (by setting the RECSMALL bit in the MODER register to 1), then the SF bit is set to 1 in the Rx BD when a short frame appears. Minimum length is defined in the PACKETLEN register in the Ethernet IP Core Specification.

Rx Big Frames (ReceivedPacketTooBig)

By default the reception of the big frames is switched off. If frame that is bigger than the maximum frame specified in the PACKETLEN register (See the Ethernet IP Core Specification) is received, then frame reception is automatically stopped at the maximum value (no big frame status is written anywhere). If reception of the big frames is enabled (See HUGEN bit in the MODER register in the Ethernet IP Core Specification), then the TL bit is set in the Rx BD when packet bigger then the maximum size is received.

Rx Dribble Nibble (DribbleNibble)

DN bit is set in the Rx BD when an extra nibble is received as a part of the frame (frame is not byte aligned). CRC error occurs at the same time, so both errors are simultaneously reported.

Tx Retry Count (RetryCntLatched)

After every frame is transmitted the number of retries is written to the RTRY field of the Tx BD. The retry count gives information about that how many times transmitter retried before successfully transmitting a frame.

Tx Retry Limit (RetryLimit)

When a number of re-transmission attempts is bigger then specified in the COLLCONF register (see Ethernet IP Core Specification B), frame transmission is aborted and bit RL is set in the Tx BD.

Tx Late Collision (LateCollLatched)

Late collision should never occur. If it occurs during the frame transmission, the transmission is aborted and LC status is written to the associated Tx BD. See COLLVALID

field of the COLLCONF register (Ethernet IP Core Specification) for more information on late collision.

Tx Defer (DeferLatched)

When frame was deferred before being sent successfully (i.e. the transmitter had to wait for Carrier Sense before sending because the line was busy), the DF bit is set in the associated Tx BD. This is not a collision indication. Collisions are indicated in RTRY.

Tx Carrier Sense Lost (CarrierSenseLost)

When Carrier Sense is lost during a frame transmission, bit CS is set in the associated Tx BD. Status is written after the frame is sent.

Following statuses are not part of the Status Module. They are generated in the module and used in the Tx and Rx BD.

Tx Underrun (UnderRun)

Underrun is detected in the WISHBONE module and reported in the Tx BD after frame transmission is aborted due to the underrun. This means that the host was not able to provide data is being transmitted on time. This is not a normal condition and should never happen.

Rx Overrun (OverRun)

Overrun is detected in the WISHBONE module and reported in the Rx BD. When Overrun status is set, It means that the host was not able to store received data to the memory on time and Rx FIFO overrun happened. Some of the data was lost.

Rx Miss (Miss)

When Ethernet MAC is configured to accept all frames regardless of their destination address (PRO bit is set in the MODER register, see Ethernet IP Core Specification), MISS bit tells if a received frame contains a valid address or not.

Additionally following signals are generated in the status module:

- **ReceivedLengthOK** reports when the received frame has a valid length

- **ReceiveEnd** reports the end of the reception. This signal is used in the control module for resetting several flip-flops and setting the pause timer.

A.6.6 Description of the Registers module (eth_registers.v)

Functionality of registers is described in the Ethernet IP Core Specification.

Although all registers are described as 32-bit registers, only the actually needed width is used. Other bits are fixed to zero (ignored on write and read as zero). Each register is instantiated with two parameters, width and reset value. Reset value defines whether register clears its value to zero or set to some predefined value after the reset.

Description of the eth_register module (eth_register.v)

This module contains one single register. The width of the register and its reset value are defined with two parameters:

- WIDTH
- RESET_VALUE.

A.6.7 Description of the WISHBONE module (eth_wishbone.v)

This module has multiple functions:

- It is the interface between the Ethernet Core and other devices (memory, host). Two WISHBONE interfaces (slave and master) are used for this manner.
- Contains buffer descriptors (in the internal RAM).
- Contains receive and transmit FIFO.
- Contains synchronisation logic for signals that spread through different clock domains.
- Transmit related function that reads TX BD and then starts WISHBONE master interface, fills the TX FIFO and then starts the transmission. At the end it writes status to the related TX BD.
- Receive related function that reads RX BD, assembles incoming bytes to words and then writes them to the RX FIFO. They are then written to the memory through the WISHBONE master interface. At the end it writes status to the related RX BD.

WISHBONE Slave Interface

Ethernet registers and buffer descriptors (BD) are all accessed through the same WISHBONE Slave Interface. Registers are located in the eth_registers module, while BDs are saved in the internal RAM within the eth_wishbone module. Selection between registers and BD accesses is done in the eth_top module. This means that all accesses that reach eth_wishbone module are meant for buffer descriptors (See following Buffer Descriptor section for more details). All output signals (from slave WISHBONE interface) can be registered or not. Selection is done with ETH_REGISTERED_OUTPUTS define in the eth_defines.v file.

WISHBONE Master Interface

The Ethernet core uses WISHBONE **master** interface for accessing the memory space where the buffers (data) are stored. Both, the receiver and the transmitter access data through the same WISHBONE master interface. For this purposes a state machine is build. The state machine multiplexes access from TX and RX modules (See MasterWbTX and MasterWbRX signals). Following signals are used in the state machine:

- MasterWbTX
- MasterWbRX
- ReadTxDataFromMemory_2
- WriteRxDataToMemory
- MasterAccessFinished
- cyc_cleared
- tx_burst
- rx_burst

When a Receiver receives data from the Ethernet and needs to store it to the memory, it asserts the **WriteRxDataToMemory** signal. Write access can start immediately or is delayed (depending if another access is already in progress, the type of the previous access and number of requested accesses). **MasterWbRX** is set to 1 when receiver uses the WISHBONE bus.

When a Transmitter needs to send data, it reads the data from the memory. **ReadTxDataFromMemory_2** is asserted when transmitter needs data from the memory. Read access can start immediately or is delayed (depending if another access is already in progress, the type of the previous access and number of requested accesses). **MasterWbTX** is set to 1 when transmitter uses the WISHBONE bus.

Every WISHBONE access is finished when slave asserts acknowledge or error signal. Both signals are joined together in **MasterAccessFinished**signal.

After every access, **m_wb_cyc_o** signal must be cleared to zero because of the traffic COP limitations. When there are two consecutive single accesses performed one after another, state machine goes to the “temporary idle” state where signal **cyc_cleared**is set and **m_wb_cyc_o**cleared to zero. After that a normal read or write operation starts. Both, single accesses and burst accesses are supported.

Accesses to/from addresses that are not word-aligned are supported.

When transmitter needs to send data that is stored in the memory at non-aligned address, following procedure is used:

Pointer to the TX buffer is stored to three different registers: **TxPointerMSB**, **TxPointerLSB** and **TxPointerLSB_rst**. TxPointerMSB is used for accessing the word-aligned memory. After every WISHBONE access TxPointerMSB is incremented and points to the next word in the memory. TxPointerLSB bits remain unchanged during the whole operation of packet sending. Since word accesses are performed, valid data does not necessarily start at byte 0 (could be byte 0, 1, 2 or 3). TxPointerLSB is used only at the beginning (when accessing the first data word) for proper selection of the start byte (TxData and TxByteCnt signals depend on it). After the read access, TxLength needs to be decremented for the number of the valid bytes (1 to 4). After the first read all bytes are valid so this two bits are reset to zero. For this reason TxPointerLSB_rst is used. This signal is the same as TxPointerLSB except that it resets to zero after the first read access.

When receiver need to store data to the memory at word-unaligned address, the following procedure is used:

Buffer descriptor pointer is stored to two different registers: **RxPointerMSB** and **RxPointerLSB_rst**. Accesses are always performed to word-aligned locations. For that reason the RxPointerMSB with two LSB bits fixed to zero are used. Byte select signals (**RxByteSel**) are used for solving the alignment problem, i.e., If RxPointer is 0x1233, then word access to 0x1230 is performed and RxByteSel is set to 0x1. RxPointerLSB_rst signal is used for **RxByteSel**, **RxByteCnt**, **RxValidBytes** and **RxDataLatched1** signals generation. RxByteSel is used as byte select signal when writing data to the memory through the wishbone interface. After the first write access, RxPointerLSB_rst is reset to zero and all byte selects (RxByteSel) become valid (only word accesses are performed).

RxByteCnt counts bytes within the word. It is used for proper latching of the input data, setting the conditions when to write data to the RX FIFO and to mark when the last byte is received through the Ethernet. **RxValidBytes** marks how many bytes are valid within the last word that is written to the memory.

Note: Even when not all bytes are valid when writing the last word to the memory, full word is written (invalid bytes are written as zeros).

Tx and Rx Buffer Descriptors

Buffer descriptors are located in the internal RAM at addresses between 0x400 and 0x7ff. Each BD is 8 bytes long (4 bytes for status and 4 bytes for pointer). Access to buffer descriptors is only possible when Ethernet MAC Controller is not in reset (See RST bit in the MODER register). As soon as the READY bit is set in the TX BD (READY bit in the RX BD), descriptor cannot be changed until transmitter clears that bit to zero (receiver). There are totally 128 buffer descriptors that can be used for both, transmit (TX) or receive (RX). Number of TX BD is defined in the TX_BD_NUM register. The rest are used for RX BD.

Example:

If value 0x32 is written in the TX_BD_NUM register, it means that there are 50 TX BD and 78 RX BD, i.e 128-50.

Tx BDs are accessible between 0x400 and 0x58c (8 x 0x32 + 0x400 - 4).

Rx BDs are accessible between 0x590 (8 x 0x32 + 0x400) and 0x7fc.

For detailed description of the buffer descriptors, please read the Buffer Descriptors (BD) section of the Ethernet IP Core Specification.

Single port RAM is used for buffer descriptors (smaller). Three devices can access RAM:

- Host through the WISHBONE slave interface
- Transmitter
- Receiver

Smart access multiplexing is done with a state machine (see generation of the WbEn, RxEn and TxEn signals). Multiplexing depends on the **RxEn_needed** and **TxEn_needed** signals.

RxEn_needed informs the state machine that the receiver needs to access a buffer descriptor in the RAM (needs to write a status (after receiving a frame) to it or needs an empty buffer descriptor to start with the reception).

After the reset **RxBDRead** is set to 1 and **RxBDReady** is set to zero. This means that there is a need to read an empty buffer descriptor from the RAM (signal RxEn_needed is set to 1). A read cycle to the **RxBDAddress** is started. If a BD that is not mark as empty is read, the same procedure is repeated. As soon as a BD that is marked as empty (bit EMPTY set to 1) is read, a pointer related to the same BD is needed. Another read is performed to the address where pointer is stored (**RxBDAddress** + **RxPointerRead**). After that there is no need for receiver to read the BDs and signal RxEn_needed is cleared to zero with **RxPointerRead** signal. Reception of the frame starts automatically. When a frame is received, signal **ShiftEnded** is set to 1. This signal clears RxBDReady signal, which then sets RxEn_needed to 1. Status is written to the related receive BD, address is incremented and read to the next BD is started.

TxEn_needed tells to the state machine there is a need that transmitter accesses the buffer descriptors in RAM. Operation of the TX BD is very similar to the operation of the Rx BD. In this case used signals are **TxBDRead**, **TxBDReady**, **TxPointerRead**, **TxStatusWrite**.

Tx and Rx FIFO

Both, TX and RX sides have FIFO-s. Defines related to the FIFO-s are in the eth_defines.v file:

- TX_FIFO_CNT_WIDTH, TX_FIFO_DEPTH, TX_FIFO_DATA_WIDTH for TX FIFO
- RX_FIFO_CNT_WIDTH, RX_FIFO_DEPTH, RX_FIFO_DATA_WIDTH for RX FIFO

Currently both FIFO-s are 16-words deep.

After the TX BD is read (both status and pointer), data is read from the memory through the master Wishbone interface and stored to the TX FIFO. Actual transmission starts as soon as the TX FIFO is full (to keep the possibility of the underruns as low as possible). When there is space for at least one word in the FIFO, another read is performed.

After the RX BD is read (both status and pointer) and there is some incoming data in the FIFO (at least one word), write to the memory is immediately performed. Reception of the next frame is possible after all data is written to the memory (FIFO is empty).

Synchronisation Logic

Typical approach was that at least two flip-flops were used when crossing different clock domains. Those signals that crossed clock domains and were available long time before it's actual use were not synchronized.

A.7 Ethernet MAC IP Core Testbench

A.7.1 Overview

Ethernet MAC IP Core testbench consists of a whole environment for testing Ethernet MAC IP Core, including Ethernet PHY model, WISHBONE bus models with bus monitors and test cases, which use those models to stimulate transactions through the Ethernet. Those transactions are checked in many different modes.

A.7.2 Testbench File Hierarchy

The hierarchy of modules in the Testbench of the Ethernet MAC IP Core is shown here with file tree. Each file here implements one module in a hierarchy. Source files of the Testbench are in the **ethernet\bench\verilog** sub-directory.

tb_ethernet.v

. **eth_phy.v**

. **wb_bus_mon.v**

. **wb_master_behavioral.v**

. . **wb_master32.v**

. **wb_slave_behavioral.v**

. **wb_model_defines.v**

. **tb_eth_defines.v**

. **eth_phy_defines.v**

A.7.3 Testbench Module Hierarchy

Module hierarchy is shown in detail in the following picture. Description of modules and their connections in section A.8, Description of Testbench Modules.

A.8 Description of Testbench Modules

The module tb_ethernet.v is used as testing environment and it incorporates beside all test sub-modules, functions and tasks also Unit Under Test (Ethernet MAC IP Core). Description of tasks is covered in chapter Description of Test-cases, while all test sub-modules are described in the following chapters.

A.8.1 Description of Ethernet PHY module

Ethernet PHY module simulates simplified Intel LXT971A PHY chip.

Ethernet PHY provides two clock signals to the Ethernet MAC Core: transmit clock (mtx_clk_o) and receive clock (mrx_clk_o). Depending on the control bits, TX and RX clock operate at 2.5 MHz for 10 Mbps operation or 25 MHz for 100 Mbps operation (only bit [13] is used for clock frequency setting). TX and RX clock signals are not synchronous.

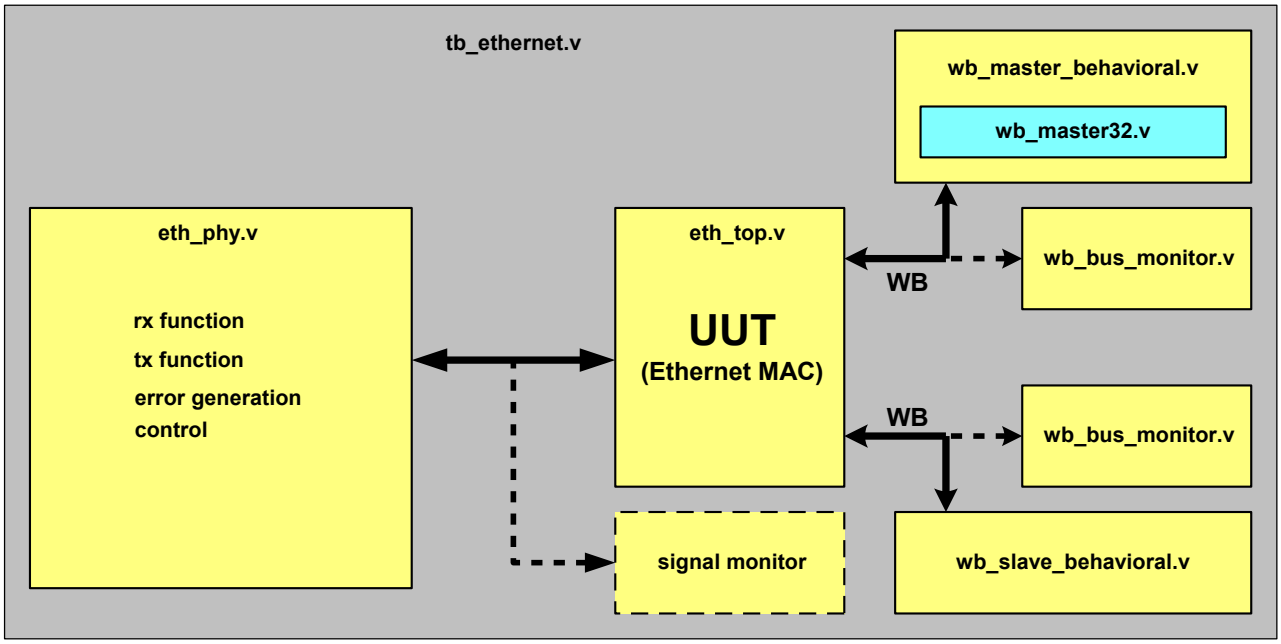


Figure A.4 Test Bench Module Hierarchy

When Ethernet link is not up, RX clock has a random frequency between 2 MHz and 40 MHz.

PHY has an MIIM interface, which is connected to the Ethernet core. All transactions are monitored and every error/warning reported. Besides that PHY has several registers implemented in it (Control, Status and two Identification registers).

PHY provides carrier sense and collision signals. Both signals can be set through several tasks.

When transmitting data (PHY is receiving data), PHY controls the protocol (preamble, sfd, writes length and data to its memory).

When PHY sends data to the Ethernet MAC, it can generate various preambles (different length, wrong preamble). It takes data from its memory. Testbench needs to write data to PHY's memory before PHY can start with transmission.

A.8.2 Description of WB sub-modules

wb_bus_monitor submodule

The module **wb_bus_monitor.v** monitors the WB Bus and tries to see WB Protocol Errors. There are two point-to-point WB buses:

- WB master from Ethernet MAC IP Core that goes to the WB Slave Behavioural unit (for writing and reading data)

- WB slave from WB Master Behavioural unit to the Ethernet MAC IP Core (used for accessing registers and buffer descriptors)

There are also two WB bus monitors, one for each WB bus.

wb_master_behavioral sub-module

The module **wb_master_behavioral.v** is used to initiates WB cycles to WB Slave in the Ethernet MAC IP Core. That is controlled by top-level. This module also includes a sub-module **wb_master32.**v, which is used to generate proper WB cycles. The length and type of each cycle is controlled by **wb_master_behavioral.v** module. This module also incorporates a block of SRAM.

wb_slave_behavioral sub-module

The module **wb_slave_behavioral.v** responds to cycles initiated by WB Master in the Ethernet MAC IP Core. When to respond and a type of cycle termination is controlled by top-level. This module also incorporates a block of SRAM.

A.9 Description of Testcases

There are some tasks not used as testcases (e.g. clear_memories, hard_reset, reset_mac, reset_mii), but are significant for proper working of testbench. All Testcases are in tb_ethernet.v file and are sometimes combined with more tasks or are just a part of one task. Some system parameters (wbm_init_waits, wbm_subseq_waits, etc.) are used with all possible combinations while running all testcases (most of testcases are used more than one time). In the following chapters basic descriptions for groups of testcases are presented. Each testcase has its name with a test type meaning. For deeper explanations of testcases see the comments in the tb_ethernet.v file. Testbench also provide log files as a result of all tests into ethernet/sim/rtl_sim/nc_sim/log directory. File eth_tb.log has all testcases results written as SUCCESSFULL or FAIL. Files eth_tb_wb_m_mon.log, eth_tb_wb_s_mon.log, eth_tb_phy.log report any wrong and suspicious activities on both WB buses and PHY signals.

A.9.1 Description of MAC Registers and BD Tests

The MAC Registers and Buffer Descriptors tests (**test_access_to_mac_reg**):

The following test cases are for testing Ethernet MAC internal registers:

- Walking 1 with single cycles across MAC registers.

- Test maximum register values and register values after writing inverse reset values and hard reset of the MAC.

The following test cases are for testing Ethernet MAC buffer descriptors:

- Walking 1 with single cycles across MAC buffer descriptors.
- Test buffer descriptors. RAM preserves values after hard reset of the MAC and resetting the logic.

A.9.2 Description of MIIM Module Tests

There are several tests for testing MII Management module:

- Test clock divider of MII management module with all possible frequencies.
- Test various readings from ‘real’ PHY registers.
- Test various writings to ‘real’ PHY registers (control and non-writable registers).
- Test reset PHY through MII management module.
- Test ‘walking one’ across PHY address (with and without preamble).
- Test ‘walking one’ across PHY’s register address (with and without preamble).
- Test ‘walking one’ across PHY’s data (with and without preamble).
- Test reading from PHY with wrong PHY address (host reading high ‘z’ data).
- Test writing to PHY with wrong PHY address and reading from correct one.
- Test sliding stop scan command immediately after read request (with and without preamble).
- Test sliding stop scan command immediately after write request (with and without preamble).
- Test BUSY and NVALID status durations during write (with and without preamble).
- Test BUSY and NVALID status durations during write (with and without preamble).
- Test BUSY and NVAILD status durations during scan (with and without preamble).
- Test scan status from PHY with detecting LINKFAIL bit (with and without preamble).
- Test scan status from PHY with sliding LINKFAIL bit (with and without preamble).
- Test sliding stop scan command immediately after scan request (with and without preamble).
- Test sliding stop scan command after 2^{nd} scan (with and without preamble).

ANNEX B
Les spécifications du Ethernet IP CORE

Ethernet IP Core Specification

Author: Igor Mohor

Revised by Amer Al-Canaan 2014

DATE: April 27, 2014 Rev. 1.20

Revision History

Rev.	Date	Author	Description
0.1	13/03/01	Igor Mohor	First Draft
0.2	17/03/01	Igor Mohor	MDC clock divider changed. Instead of the clock select bits CLKS[2:0] the clock divider bits CLKDIV[7:0] are used.
1.0	21/03/01	Igor Mohor	MII module completed. Revision changed to 1.0 due to cvs demands.
1.1	16/04/01	IM	DMA support and buffer descriptors added.
1.2	24/05/01	IM	Registers revised.
1.3	05/06/01	IM	Status is written to the status registers. DMA channels 2 and 3 are not used any more. Figures that are implementation specific removed from the document.
1.4	03/07/01	IM	COLLCONF register changed bit width. BCKPRESS and BCKPNBEN bit removed from MODER. LOOPBCK added.
1.5	21/07/01	IM	Signal RD0_O (Restart Descriptor for channel 0) added. Per packet CRC, BD changed.
1.6	03/12/01	IM	BD Section rewritten.
1.7	05/12/01	IM	TX_BD_NUM register used instead of RX_BD_BASE_ADDR register.
1.8	07/01/02	IM	Minor typos fixed
1.9	30/01/02	IM	RST bit in MODER register is 1 after reset, initial collision window changed.
1.10	18/02/02	IM	Address recognition system added. Buffer

			Descriptors changed. DMA section changed. Ports changed.
1.11	02/03/02	IM	Typos fixed, INT_SOURCE and INT_MASK registers changed.
1.12	15/03/02	IM	TX_BD_NUM, MAC_ADDR0 and MAC_ADDR1 register description changed.
1.13	15/04/02	Jeanne Wiegelmann	Document revised.
1.14	14/05/02	IM	Minor typos fixed.
1.15	14/08/02	IM	LINKFAIL and NVALID bit description changed in the MIISTATUS register. TX_BD_NUM changed. External DMA support removed from the document.
1.16	04/09/02	IM	RX_BD_NUM changed to TX_BD_NUM in the register table. MIIRX_DATA bits changed to read only. MIIMRST bit in MIIMODER register moved from bit [10] to bit [9]. Description of the RXEN and TXEN bits in MODER register is modified. Description of the CTRLMODER and INT_SOURCE register changed.
1.17	14/11/02	IM	Few minor changes in the Tx BD, Rx BD and Host Interface sections. Bit description in the INT_SOURCE register improved. Frame reception section fixed. Minimum tx and rx length defined in the Frame Transmission and Frame Reception sections. RST bit in MODER removed.
1.18	22/11/02	IM	MIIMRST (Reset of the MIIM module) not used any more in the MIIMODER register. Control Frame bit (CF) added to the RX buffer descriptor. Control frame detection section updated.
1.19	27/11/02	IM	Control frame detection section improved.
1.20	04/04/014	Amer Al-Canaan	General revisions and updates specially to the TXBD and RXBD sections.

B.1 Introduction

The Ethernet IP Core consists of five modules:

- The MAC (Media Access Control) module, formed by transmit, receive, and control module
- The MII (Media Independent Interface) Management module
- The Host Interface

The Ethernet IP Core is capable of operating at 10 or 100 Mbps for Ethernet and Fast Ethernet applications. An external PHY is needed for the complete Ethernet solution.

B.2 IO Ports

B.2.1 Ethernet Core IO ports

The Ethernet IP Core uses three types of signals to connect to media:

- WISHBONE signals to connect to the Host Interface.
- MII Management signals to connect to the PHY.
- Reset signals (for resetting different parts of the Ethernet IP Core.

B.2.2 Host Interface Ports

The table below contains the common ports connecting the Ethernet IP Core to the Host Interface. The Host Interface is WISHBONE Rev. B compliant.

All signals listed below are active HIGH, unless otherwise noted. Signal direction is with respect to the Ethernet IP Core.

Port	Width	I or O	Description
CLK_I	1	I	Clock Input
RST_I	1	I	Reset Input
ADDR_I	32	I	Address Input
DATA_I	32	I	Data Input
DATA_O	32	O	Data Output
SEL_I	4	I	Select Input Array Indicates which bytes are valid on the data bus.

			Whenever this signal is not 1111b during a valid access, the ERR_O is asserted.
WE_I	1	I	Write Input Indicates a Write Cycle when asserted high or a Read Cycle when asserted low.
STB_I	1	I	Strobe Input Indicates the beginning of a valid transfer cycle.
CYC_I	1	I	Cycle Input Indicates that a valid bus cycle is in progress.
ACK_O	1	O	Acknowledgment Output Indicates a normal Cycle termination.
ERR_O	1	O	Error Acknowledgment Output Indicates an abnormal cycle termination.
INTA_O	1	O	Interrupt Output A.
M_ADDR_O	32	O	Master Address Output M_DATA_I
M_DATA_I	32	I	Master Data Input
M_DATA_O	32	O	Master Data Output
M_SEL_O	4	I	Master Select Output Array Indicates which bytes are valid on the data bus. Whenever this signal is not 1111b during a valid access, the ERR_I is asserted.
M_WE_O	1	O	Master Write Output Indicates a Write Cycle when asserted high or a Read Cycle when asserted low.
M_STB_O	1	O	Master Strobe Output Indicates the beginning of a valid transfer cycle.
M_CYC_O	1	O	M_CYC_O Indicates that a valid bus cycle is in progress.
M_ACK_I	1	I	Master Acknowledgment Input Master Acknowledgment Input
M_ERR_I	1	I	Error Acknowledgment Input Indicates an abnormal cycle termination.

Table B.2 Host Interface ports

B.2.3 PHY Interface ports

The table below contains the ports connecting the Ethernet IP Core to the PHY Interface. All signals listed below are active HIGH, unless otherwise noted. Signal direction is with respect to the Ethernet IP Core.

Port	Width	I/O	Description
MTxClk	1	I	Transmit Nibble or Symbol Clock. The PHY provides the MTxClk signal. It operates at a frequency of 25 MHz (100 Mbps) or 2.5 MHz (10 Mbps). The clock is used as a timing reference for the transfer of MTxD[3:0], MtxEn, and MTxErr.
MTxD [3:0]	4	O	Transmit Data Nibble. Signals are the transmit data nibbles. They are synchronized to the rising edge of MTxClk. When MTxEn is asserted, PHY accepts the MTxD.
MTxEn	1	O	Transmit Enable. When asserted, this signal indicates to the PHY that MTxD[3:0] the data are valid and the transmission can start. The transmission starts with the first nibble of the preamble. The signal remains asserted until all nibbles to be transmitted are presented to the PHY. It is deasserted prior to the first MTxClk, following the final nibble of a frame.
MTxErr	1	O	Transmit Coding Error. When asserted for one MTxClk clock period while MTxEn is also asserted, this signal causes the PHY to transmit one or more symbols that are not part of the valid data or delimiter set somewhere in the frame being transmitted to indicate that there has been a transmit coding error.
MRxClk	1	I	Receive Nibble or Symbol Clock. The PHY provides the MRxClk signal, which operates at 25 MHz (100 Mbps) or 2.5 MHz (10 Mbps). The clock is used as a timing reference for the reception of MRxD[3:0], MRxDV, and MRxErr.
MRxDV	1	I	Receive Data Valid. The PHY asserts this signal to indicate to the Rx MAC that it is presenting the valid nibbles on the MRxD[3:0] signals. The signal is asserted synchronously to the MRxClk. MRxDV is asserted from the first recovered nibble of the frame to the final recovered nibble. It is then deasserted prior to the first MRxClk that follows the final nibble.

MRxD [3:0]	4	I	Receive Data Nibble. These signals are the receive data nibble. They are synchronized to the rising edge of MRxClk. When MRxDV is asserted, the PHY sends a data nibble to the Rx MAC. For a correctly interpreted frame, seven bytes of a preamble and a completely formed SFD must be passed across the interface.
MRxErr	1	I	Receive Error. The PHY asserts this signal to indicate to the Rx MAC that a media error was detected during the transmission of the current frame. MRxErr is synchronous to the MRxClk and is asserted for one or more MRxClk clock periods and then deasserted.
MColl	1	I	Collision Detected. The PHY asynchronously asserts the collision signal MColl after the collision has been detected on the media. When deasserted, no collision is detected on the media.
MCrS	1	I	Carrier Sense. The PHY asynchronously asserts the carrier sense MCrS signal after the medium is detected in a non-idle state. When deasserted, this signal indicates that the media is in an idle state (and the transmission can start).
MDC	1	O	Management Data Clock. This is a clock for the MDIO serial data channel.
MDIO	1	I/O	Management Data Input/Output. Bi-directional serial data channel for PHY/STA communication.

Table B.3 PHY Interface ports

B.3 Registers

This section describes all base, control, and status registers inside the Ethernet IP Core. The Address field indicates a relative address in hexadecimal. Width specifies the number of bits in the register, and Access specifies the valid access types to that register. RW stands for read and write access, R for read-only access.

Name	Address	Width	Access	Description
MODER	0x00	17	RW	Mode Register
INT_SOURCE	0x01	7	RW	Interrupt Source Register

INT_MASK	0x02	7	RW	Interrupt Mask Register
IPGT	0x03	7	RW	Back to Back Inter Packet Gap Register
IPGR1	0x04	7	RW	Non Back to Back Inter Packet Gap Register 1
IPGR2	0x05	7	RW	Non Back to Back Inter Packet Gap Register 2
PACKETLEN	0x06	32	RW	Packet Length (minimum and maximum) Register
COLLCONF	0x07	10	RW	Collision and Retry Configuration
TX_BD_NUM	0x08	8	RW	Transmit Buffer Descriptor Number
CTRLMODER	0x09	3	RW	Control Module Mode Register
MIIMODER	0x0A	9	RW	MII Mode Register
MIICOMMAND	0x0B	3	RW	MII Commend Register
MIIADDRESS	0x0C	10	RW	MII Address Register Contains the PHY address and the register within the PHY address
MIITX_DATA	0x0D	16	RW	MII Transmit Data Width Data to be transmitted to the PHY
MIIRX_DATA	0x0E	16	RW	MII Receive Data Width The data received from the PHY
MIISTATUS	0x0F	3	RW	MII Status Register
MAC_ADDR0	0x10	32	RW	MAC Individual Address0 The LSB four bytes of the individual address are written to this register.
MAC_ADDR1	0x11	16	RW	MAC Individual Address1 The MSB two bytes of the individual address are written to this register
ETH_HASH0_ADR	0x12	32	RW	HASH0 Register
ETH_HASH1_ADR	0x13	32	RW	HASH1 Register
ETH_TX_CTRL	0x14	17	RW	Transmit Control Register
ETH_RX_CTRL	0x15	16	RW	Receive control address
ETH_DBG	0x16	16	RW	Receive control address

Table B.4 Register List

B.3.1 MODER (Mode Register)

Bit #	Access	Description
31-17		Reserved
16	RW	RECSMALL - Receive Small Packets RECSMALL - Receive Small Packets 1 = Packets smaller than MINFL are accepted.
15	RW	PAD - Padding enabled 0 = Do not add pads to short frames. 1 = Add pads to short frames (until the minimum frame length is equal to MINFL).
14	RW	HUGEN - Huge Packets Enable 0 = The maximum frame length is MAXFL. All additional bytes are discarded. 1 = Frames up 64 KB are transmitted.
13	RW	CRCEN - CRC Enable 0 = Tx MAC does not append the CRC (passed frames already contain the CRC). 1 = Tx MAC appends the CRC to every frame.
12	RW	DLYCRCEN - Delayed CRC Enabled 0 = Normal operation (CRC calculation starts immediately after the SFD). 1 = CRC calculation starts 4 bytes after the SFD.
11		Reserved
10	RW	FULLD - Full Duplex 0 = Half duplex mode. 1 = Full duplex mode.
9	RW	EXDFREN - Excess Defer Enabled 0 = When the excessive deferral limit is reached, a packet is aborted. 1 = MAC waits for the carrier indefinitely.
8	RW	NOBCKOF - No Back off 0 = Normal operation (a binary exponential back off algorithm is used). 1 = Tx MAC starts retransmitting immediately after the collision.
7	RW	LOOPBCK - Loop Back 0 = Normal operation. 1 = TX is looped back to the RX.

6	RW	IFG - Interframe Gap for Incoming frames 0 = Normal operation (minimum IFG is required for a frame to be accepted). 1 = All frames are accepted regardless of the IFG.
5	RW	PRO - Promiscuous 0 = Check the destination address of the incoming frames. 1 = Receive the frame regardless of its address.
4	RW	IAM - Individual Address Mode 0 = Normal operation (physical address is checked when the frame is received. 1 = The individual hash table is used to check all individual addresses received.
3	RW	BRO - Broadcast Address 0 = Receive all frames containing the broadcast address. 1 = Reject all frames containing the broadcast address unless the PRO bit = 1.
2	RW	NOPRE - No Preamble 0 = Normal operation (7-byte preamble). 1 = No preamble is sent.
1	RW	TXEN - Transmit Enable 0 = Transmit is disabled. 1 = Transmit is enabled. If the value, written to the TX_BD_NUM register, is equal to 0x0 (zero buffer descriptors are used), then the transmitter is automatically disabled regardless of the TXEN bit.
0	RW	RXEN - Receive Enable 0 = Receive is disabled. 1 = Receive is enabled. If the value, written to the TX_BD_NUM register, is equal to 0x80 (all buffer descriptors are used for transmit buffer descriptors, so there is no receive BD), then receiver is automatically disabled regardless of the RXEN bit.

Table B.5 MODE Register

Reset Value: MODER: 0000A000h
NOTE: Registers should not be changed after the TXEN or RXEN bits is set.

B.3.2 INT_SOURCE (Interrupt Source Register)

Bit #	Access	Description
31-7		Reserved
6	RW	RXC - Receive Control Frame This bit indicates that the control frame was received. It is cleared by writing 1 to it. Bit RXFLOW (CTRLMODER register) must be set to 1 in order to get the RXC bit set
5	RW	TXC - Transmit Control Frame This bit indicates that a control frame was transmitted. It is cleared by writing 1 to it. Bit TXFLOW (CTRLMODER register) must be set to 1 in order to get the TXC bit set.
4	RW	BUSY - Busy This bit indicates that a buffer was received and discarded due to a lack of buffers. It is cleared by writing 1 to it. This bit appears regardless of the IRQ bits in the Receive or Transmit Buffer Descriptors.
3	RW	RXE - Receive Error This bit indicates that an error occurred while receiving data. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Receive Buffer Descriptor.
2	RW	RXB - Receive Frame This bit indicates that a frame was received. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Receive Buffer Descriptor. If a control frame is received, then RXC bit is set instead of the RXB bit. (See CTRLMODER description in section B.4.4 for more details.)
1	RW	TXE - Transmit Error This bit indicates that a buffer was not transmitted due to a transmit error. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Receive Buffer Descriptor. This bit appears only when IRQ bit is set in the Transmit Buffer Descriptor.
0	RW	TXB - Transmit Buffer This bit indicates that a buffer has been transmitted. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Transmit Buffer Descriptor.

Table B.6 INT_SOURCE Register

Reset Value: INT_SOURCE: 00000000h

B.3.3 INT_MASK (Interrupt Mask Register)

Bit #	Access	Description
31-7		Reserved
6	RW	RXC_M - Receive Control Frame Mask 0 = Event masked 1 = Event causes an interrupt
5	RW	TXC_M - Transmit Control Frame Mask 0 = Event masked 1 = Event causes an interrupt
4	RW	BUSY_M - Busy Mask 0 = Event masked 1 = Event causes an interrupt
3	RW	RXE_M - Receive Error Mask 0 = Event masked 1 = Event causes an interrupt
2	RW	RXF_M - Receive Frame Mask 0 = Event masked 1 = Event causes an interrupt
1	RW	TXE_M - Transmit Error Mask 0 = Event masked 1 = Event causes an interrupt
0	RW	TXB_M - Transmit Buffer Mask 0 = Event masked 1 = Event causes an interrupt

Table B.7 INT_MASK Register

Reset Value:

INT_MASK: 00000000h

B.3.4 IPGT (Back to Back Inter Packet Gap Register)

Bit #	Access	Description
31-7		Reserved
6-0	RW	IPGT - Back to Back Inter Packet Gap Full Duplex: The recommended value is 0x15, which equals 0.96 μs IPG (100 Mbps) or 9.6 μs (10 Mbps). The desired period in nibble times minus 6 should be written to the register. Half Duplex: The recommended value and default is 0x12, which equals 0.96 μs IPG (100 Mbps) or 9.6 μs (10 Mbps). The desired period in nibble times minus 3 should be written to the register.

Table B.8 IPGT Register

Reset Value:

IPGT: 00000012h

B.3.5 IPGR1 (Non Back to Back Inter Packet Gap Register 1)

Bit #	Access	Description
31-7		Reserved
6-0	RW	IPGR1 – Non Back to Back Inter Packet Gap 1 When a carrier sense appears within the IPGR1 window, Tx MAC defers and the IPGR counter is reset. When a carrier sense appears later than the IPGR1 window, the IPGR counter continues counting. The recommended and default value for this register is 0xC. It must be within the range [0,IPGR2].

Table B.9 IPGR1 Register

Reset Value:

IPGR1: 0000000Ch

B.4 IPGR2 (Non Back to Back Inter Packet Gap Register 2)

Bit #	Access	Description
31-7		Reserved
6-0	RW	IPGR2 - Non Back to Back Inter Packet Gap 2 The recommended and default value is 0x12, which equals to 0.96 μs IPG (100 Mbit/s) or 9.6 μ s (10 Mbit/s).

Table B.10 IPGR2 Register

Reset Value:

IPGR2: 00000012h

B.4.1 PACKETLEN (Packet Length Register)

Bit #	Access	Description
31-16	RW	MINFL – Minimum Frame Length The minimum Ethernet packet is 64 bytes long. If a reception of smaller frames is needed, assert the RECSMALL bit (in the mode register MODER) or change the value of this register. To transmit small packets, assert the PAD bit or the MINFL value (see the PAD bit description in the MODER register).
15-0	RW	MAXFL – Maximum Frame Length The maximum Ethernet packet is 1518 bytes long. To support this and to leave some additional space for the tags, a default maximum packet length equals 1536 bytes (0x0600). If there is a need to support bigger packets, you can assert the HUGEN bit or increase the value of the MAXFL field (see the HUGEN bit description in the MODER).

Table B.11 PACKETLEN Register

Reset Value:

PACKETLEN: 00400600h

B.4.2 COLLCONF (Collision and Retry Configuration Register)

Bit #	Access	Description
31-20		Reserved
19-16	RW	MAXRET – Maximum Retry This field specifies the maximum number of consequential retransmission attempts after the collision is detected. When the maximum number has been reached, the Tx MAC reports an error and stops transmitting the current packet. According to the Ethernet standard, the MAXRET default value is set to 0xf (15).
15-6		Reserved
5-0	RW	COLLVALID – Collision Valid This field specifies a collision time window. A collision that occurs later than the time window is reported as a Late “Collision” and transmission of the current packet is aborted. The default value equals 0x3f (by default, a late collision is every collision that occurs 64 bytes (63 + 1) from the preamble).

Table B.12 COLLCONF Register

Reset Value:

COLLCONF: 000F003fh

B.4.3 TX_BD_NUM (Transmit BD Number Reg.)

Bit #	Access	Description
31:8		Reserved
7:0	RW	Transmit Buffer Descriptor (TX_BD) Number Number of the Tx BD. Number of the Rx BD equals to the (0x80 - Tx BD number). Maximum number of the Tx BD is 0x80. Values greater then 0x80 cannot be written to this register (ignored).

Table B.13 TX_BD_NUM Register

Reset Value: TX_BD_NUM: 00000040h

B.4.4 CTRLMODER (Control Module Mode Register)

Bit #	Access	Description
31-3		Reserved
2	RW	TXFLOW – Transmit Flow Control 0 = PAUSE control frames are blocked. 1 = PAUSE control frames are allowed to be sent. This bit enables the TXC bit in the INT_SOURCE register.
1	RW	RXFLOW – Receive Flow Control 0 = Received PAUSE control frames are ignored. 1 = The transmit function (Tx MAC) is blocked when a PAUSE control frame is received. This bit enables the RXC bit in the INT_SOURCE register.
0	RW	PASSALL – Pass All Receive Frames 0 = Control frames are not passed to the host. RXFLOW must be set to 1 in order to use PAUSE control frames. 1 = All received frames are passed to the host.

Table B.14 CTRLMODER Register

Reset Value: CTRLMODER: 00000000h

PASSALL	RXFLOW	Description
0	0	When a PAUSE control frame is received, nothing happens. The control frame is not stored to the memory.
0	1	When a PAUSE control frame is received, RXC interrupt is set and pause timer is updated. The control frame is not stored to the memory.
1	0	When a PAUSE control frame is received, it is stored to the as a normal-data frame. RXB interrupt is set (if the related buffer memory descriptor has an IRQ bit set to 1). RXC interrupt is not set and pause timer is not updated.
1	1	When a PAUSE control frame is received, RXC interrupt is set and pause timer is updated. Besides that the control frame is also stored to the memory as a normal data frame.

Table B.15 PASSALL and RXFLOW operation

B.4.5 MIIMODER (MII Mode Register)

Bit #	Access	Description
31-9		Reserved
8	RW	MIINOPRE – No Preamble 0 = 32-bit preamble sent 1 = No preamble send
7-0		CLKDIV – Clock Divider The field is a host clock divider factor. The host clock can be divided by an even number, greater then 1. The default value is 0x64 (100).

Table B.16 MIIMODER Register

Reset Value:

MIIMODER: 00000064h

B.4.6 MIICOMMAND (MII Command Register)

Bit #	Access	Description
31-3		Reserved
2	RW	WCTRLDATA – Write Control Data
1	RW	RSTAT – Read Status
0	RW	SCANSTAT – Scan Status

Table B.17 MIICOMMAND Register

Reset Value:

MIICOMMAND: 00000000h

NOTE: While one operation is in progress, BUSY signal, MIISTATUS (MII Status Register) in section B.4.10, is set. Next operation can be started after the previous one is finished (and BUSY signal cleared to zero).

B.4.7 MIIADDRESS (MII Address Register)

Bit #	Access	Description
31-13		Reserved
12-8	RW	RGAD – Register Address (within the PHY selected by the FIAD[4:0])
7-5		Reserved
4-0	RW	FIAD – PHY Address

Table B.18 MIIADDRESS Register

Reset Value:

MIIADDRESS: 00000000h

B.4.8 MIITX_DATA (MII Transmit Data)

Bit #	Access	Description
31-16		Reserved
15-0	RW	CTRLDATA – Control Data (data to be written to the PHY)

Table B.19 MIITX_DATA Register

Reset Value:

MIITX_DATA: 00000000h

B.4.9 MIIRX_DATA (MII Receive Data)

Bit #	Access	Description
31-16		Reserved
15-0	R	PRSD – Received Data (data read from the PHY)

Table B.20 MIIRX_DATA Register

Reset Value:

MIIRX_DATA: 00000000h

B.4.10 MIISTATUS (MII Status Register)

Bit #	Access	Description
31-3		Reserved
2	R	NVALID – Invalid 0 = The data in the MSTATUS register is valid. 1 = The data in the MSTATUS register is invalid. This bit is only valid when the scan status operation is active.
1	R	BUSY 0 = The MII is ready. 1 = The MII is busy (operation in progress).
0	R	LINKFAIL: 0 = The link is OK. 1 = The link failed. The Link fail condition occurred (now the link might be OK). Another status read gets a new status.

Table B.21 MIISTATUS Register

Reset Value:

MIISTATUS: 00000000h

B.4.11 MAC_ADDR0 (MAC Address Register 0)

Bit #	Access	Description
31-24	RW	Byte 2 of the Ethernet MAC address (individual address)
23-16	RW	Byte 3 of the Ethernet MAC address (individual address)
15-8	RW	Byte 4 of the Ethernet MAC address (individual address)
7-0	RW	Byte 5 of the Ethernet MAC address (individual address)

Table B.22 MAC_ADDR0 Register

Reset Value:

MAC_ADDR0: 00000000h

Note: When an address is transmitted, byte 0 is sent first and byte 5 last.

B.4.12 MAC_ADDR1 (MAC Address Register 1)

Bit #	Access	Description
31-16		Reserved
15-8	RW	Byte 0 of the Ethernet MAC address (individual address)
7-0	RW	Byte 1 of the Ethernet MAC address (individual address)

Table B.23 MAC_ADDR1 Register

Reset Value:

MAC_ADDR1: 00000000h

Note: When an address is transmitted, byte 0 is sent first and byte 5 last.

B.4.13 HASH0 (HASH Register 0)

Bit #	Access	Description
31-0	RW	Hash0 value

Table B.24 HASH0 Register

Reset Value:

HASH0: 00000000h

B.4.14 HASH1 (HASH Register 1)

Bit #	Access	Description
31-0	RW	Hash1 value

Table B.25 HASH1 Register

Reset Value:

HASH1: 00000000h

B.4.15 TXCTRL (Tx Control Register)

Bit #	Access	Description
31-17		Reserved
16		TXPAUSERQ – Tx Pause Request Writing 1 to this bit starts sending control frame procedure. Bit is automatically cleared to zero.
15:0	RW	TXPAUSETV – Tx Pause Timer Value The value that is send in the pause control frame.

Table B.26 Tx Control Register

Reset Value:

TXCTRL: 00000000h

B.5 Operation

This section describes the Ethernet IP Core operation.

The core consists of five modules:

- The host interface connects the Ethernet Core to the rest of the system via the WISHBONE (using DMA transfers). Registers are also part of the host interface.
- The TX Ethernet MAC performs transmit functions.
- The RX Ethernet MAC performs receive functions.
- The MAC Control Module performs full duplex flow control functions.
- The MII Management Module performs PHY control and gathers the status information from it.

All modules combined deliver full-function 10/100 Mbps Media Access Control. The Ethernet IP Core can operate in half- or full-duplex mode and is based on the CSMA/CD (Carrier Sense Multiple Access / Collision Detection) protocol.

When a station wants to transmit in half-duplex mode, it must observe the activity on the media (Carrier Sense). As soon as the media is idle (no transmission), any station can start transmitting (Multiple Access). If two or more stations are transmitting at the same time, a collision on the media is detected. All stations stop transmitting and back off for

some random time. After the back-off time, the station checks the activity on the media again. If the media is idle, it starts transmitting. All other stations wait for the current transmission to end.

In full-duplex mode, the Carrier Sense and the Collision Detect signals are ignored. The MAC Control module takes care of sending and receiving the PAUSE control frame to achieve Flow control (see the TXFLOW and RXFLOW bit description in the CTRLMODER register for more information).

The MII Management module provides a media independent interface (MII) to the external PHY. This way, the configuration and status registers of the PHY can be read from/written to.

B.6 Resetting Ethernet Core

The RST_I signal is used for resetting all sub-modules except the MIIM module. Setting the MIIMRST bit in the MIIMODER register to 1 resets the MIIM module. To reset the PHY, assert its RESET signal either through the boars system control register or by writing an appropriate bit in the PHY register.

B.7 Host Interface Operation

The host interface connects the Ethernet IP Core to the rest of the system (RISC, memory) via the WISHBONE bus. The WISHBONE serves to access the configuration registers and the memory. Currently, only DMA transfers are supported for transferring the data from/to the memory.

B.7.1 Configuration Registers

The function of the configuration registers is transparent and can be easily understood by reading the Registers section B.3.

B.7.2 Buffer Descriptors (BD)

The transmission and the reception processes are based on the descriptors. The Transmit Descriptors (TxD) are used for transmission while the Receive Descriptors (RxD) are used for reception.

The buffer descriptors are 64 bits long. The first 32 bits are reserved for length and status while the last 32 bits contain the pointer to the associated buffer (where data is stored). The Ethernet MAC core has an internal RAM that can store up to 128 BDs (for both Rx and Tx).

The internal memory saves all descriptors at addresses from 0x400 to 0x7ff (128 64bit descriptors). The transmit descriptors are located between the start address (0x400) and the address that equals the value written in the TX_BD_NUM register in section B.4.3 multiplied by 8. This register holds the number of the used Tx buffer descriptors. The receive descriptors are located between the start address (0x400), plus the address number written in the TX_BD_NUM multiplied by 8, and the descriptor end address (0x7ff).

The transmit and receive status of the packet is written to the associated buffer descriptor once its transmission/reception is finished.

For example, when the TX_BD_NUM register is initialised with 0x08 (total of 8 Tx BDs and 120 RxBDs): The TXBD address is between (as shown on figure B.1: 0x400 and 0x400 + 0x08 (0x08) - 0x04 Start address : 0x400 End address : 0x43C The RXBD address is between: 0x400 + 0x08 (0x08) and 0x7FC or (0x800 - 0x04) Start address: 0x440 End address: 0x7FC

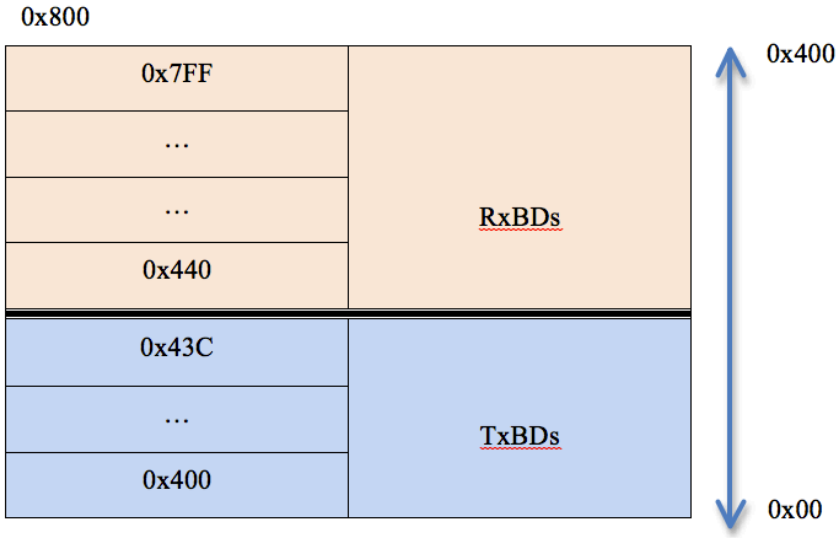


Figure B.1 Memory mapping for TX_BD_NUM = 8

NOTE : while addresses are incremented by 4 before being connected to the slave bus inputs, the first two bits are discarded on the bus so that real locations in memory are incremented by 1 inside the ethmac module. The base address 0x400 is translated internally as 0x100.

Tx Buffer Descriptors

The transmit descriptors contain information about associated buffers (length, status) and pointers to the buffers holding the relevant data.

ADDR = Offset + 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LEN															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RD	IRQ	WR	PAD	CRC	Reserved		UR	RTRY[3:0]				RL	LC	DF	CS

ADDR = Offset + 4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXPNT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXPNT															

Table B.28 Tx Buffer Descriptor (TX_BD)

Bit #	Access	Description
31-16	RW	LEN - Length Number of bytes associated with the BD to be transmitted.
15	RW	RD – Tx BD Ready 0 = The buffer associated with this buffer descriptor is not ready, and you are free to manipulate it. After the data from the associated buffer has been transmitted or after an error condition occurred, this bit is cleared to 0. 1 = The data buffer is ready for transmission or is currently being transmitted. You are not allowed to manipulate this descriptor once this bit is set.
14	RW	IRQ – Interrupt Request Enable 0 = No interrupt is generated after the transmission. 1 = When data associated with this buffer descriptor is sent, a TXB or TXE asserted (INT_SOURCE, Interrupt Source Register in section B.3.2, for more details).
13	RW	WRAP 0 = This buffer descriptor is not the last descriptor in the buffer

		descriptor table. 1 = This buffer descriptor is the last descriptor in the buffer descriptor table. After this buffer descriptor is used, the first Tx buffer descriptor in the table will be used again
12	RW	PAD– Pad Enable 0 = No pads will be added at the end of short packets. 1 = Pads will be added at the end of short packets.
11	RW	CRC – CRC Enable 0 = CRC will not be added at the end of the packet. 1 = CRC will be added at the end of the packet.
10:9		Reserved
8	RW	UR – Underrun Underrun occurred while sending this buffer.
7:4	RW	RTRY – Retry Count These bits indicate the number of retries before the frame was successfully sent.
3	RW	RL – Retransmission Limit This bit is set when the transmitter fails. (Retry Limit + 1) attempts to successfully transmit a message due to repeated collisions on the medium. The Retry Limit is set in the COLLCONF register in section B.4.2.
2	RW	LC – Late Collision Late collission occurred while sending this buffer. the transmisison is stopped and this bit is written. Late collision is defined in the COLLCONF rgister in section B.4.2.
1	RW	DF – Defer Indication
		The frame was deferred before being sent successfully, i.e. the transmitter had to wait for Carrier Sense before sending because the line was busy. This is not a collision indication. Collisions are indicated in RTRY.
0	RW	CS – Carrier Sense Lost This bit is set when Carrier Sense is lost during a frame transmission. The Ethernet controller writes CS after it finishes sending the buffer.

Table B.29 Tx Buffer Descriptor Details

Bit #	Access	Description
31-0	RW	TXPNT – Transmit Pointer This is the buffer pointer when the associated frame is stored.

Table B.30 Tx Buffer Pointer

Rx Buffer Descriptors

The receive BDs contain information about the received frames (length, status) and pointers to the buffers holding the relevant data.

ADDR = Offset + 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LEN															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	IRQ	WR	Reserved				CF	M	OR	IS	DN	TL	SF	CRC	LC

Table B.31 Rx Buffer Descriptor (RX_BD)

ADDR = Offset + 4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RXPNT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXPNT															

Table B.32 Rx Buffer Descriptor Deetails

Bit #	Access	Description
31-16	RW	LEN – Number of the received bytes associated with this BD.
15	RW	E – Empty 0 = The data buffer associated with this buffer descriptor has been filled with data or has stopped because an error occurred. The core can read or write this BD. As long as this bit is zero, this buffer descriptor won't be used. 1 = The data buffer is empty (and ready for receiving data) or currently receiving data.

14	RW	IRQ – Interrupt Request Enable 0 = No interrupt is generated after the reception. 1 = When data is received (or error occurs), an RXF interrupt will be asserted (See INT_SOURCE (Interrupt Source Register) in section B.3.2 for more details).
13	RW	WRAP 0 = This buffer descriptor is not the last descriptor in the buffer descriptor table. 1 = This buffer descriptor is the last descriptor in the buffer descriptor table. After this buffer descriptor is used, the first Rx buffer descriptor in the table will be used again.
12:9		Reserved.
8	RW	CF - Control Frame 0 = Normal data frame received 1 = Control frame received
7	RW	M - Miss 0 = The frame is received because of an address recognition hit. 1 = The frame is received because of promiscuous mode. The Ethernet controller sets M for frames that are accepted in promiscuous mode but are tagged as a miss by internal address recognition. Thus, in promiscuous mode, M determines whether a frame is destined for this station.
6	RW	OR - Overrun This bit is set when a receiver overrun occurs during frame reception.
5	RW	IS - Invalid Symbol This bit is set when the reception of an invalid symbol is detected by the PHY.
4	RW	DN - Dribble Nibble This bit is set when a received frame cannot de divided by 8 (one extra nibble has been received).
3	RW	TL - Too Long This bit is set when a received frame is too long (bigger than the value set in the PACKETLEN register (section B.4.1).
2	RW	SF - Short Frame This bit is set when a frame that is smaller than the minimum length is received (minimum length is set in the PACKETLEN register, in

		section B.4.1).
1	RW	CRC - Rx CRC Error This bit is set when a received frame contains a CRC error.
0	RW	LC - Late Collision This bit is set when a late collision occurred while receiving a frame.

Table B.33 Rx Buffer Descriptor

Bit #	Access	Description
31-0	RW	RXPNT – Receive Pointer This is the pointer to the buffer storing the associated frame.

Table B.34 Rx Buffer Pointer

B.7.3 Frame Transmission

To transmit the first frame, the RISC must do several things, namely:

- Store the frame to the memory.
- Associate the Tx BD in the Ethernet MAC core with the packet written to the memory (length, pad, crc, etc.). See section B.7.2 Buffer Descriptors (BD) for more information.
- Enable the TX part of the Ethernet Core by setting the TXEN bit to 1.

As soon as the Ethernet IP Core is enabled, it continuously reads the first BD. Immediately when the descriptor is marked as ready, the core reads the pointer to the memory storing the associated data and starts then reading data to the internal FIFO. At the moment the FIFO is full, transmission begins.

At the end of the transmission, the transmit status is written to the buffer descriptor and an interrupt might be generated (when enabled). Next, two events might occur (according to the WR bit (wrap) in the descriptor):

- If the WR bit has not been set, the BD address is incremented, the next descriptor is loaded, and the process starts all over again (if next BD is marked as ready).
- If the WR bit has been set, the first BD address (base) is loaded again. As soon as the BD is marked as ready, transmission will start.

NOTE: Only frames with length greater than 4 bytes are transmitted. Even if the BD is marked as ready, the frame is not transmitted if the length is too small.

B.7.4 Frame Reception

To receive the first frame, the RISC must do several things, namely:

- Set the receive buffer descriptor to be associated with the received packet and mark it as empty.
- Enable the Ethernet receive function by setting the RECEN bit to 1.

The Ethernet IP Core reads the Rx BD. If it is marked as empty, it starts receiving frames. The Ethernet receive function receives an incoming frame nibble per nibble. After the whole frame has been received and stored to the memory, the receive status is written to the BD. An interrupt might be generated (if enabled). Then the BD address is incremented and the next BD loaded. If the new BD is marked as empty, another frame can be received; otherwise the operation stops.

NOTE: Only frames with length greater than 4 bytes are received without an error. Smaller frames are received with a CRC error (CRC is 4-bytes long).

B.7.5 TX Ethernet MAC

The TX Ethernet MAC generates 10BASE-T/100BASE-TX transmit MII nibble data streams in response to the byte streams the transmit logic (host) supplies. It performs the required deferral and back-off algorithms, takes care of the inter-packet gap (IPG), computes the checksum (FCS), and monitors the physical media (by monitoring Carrier Sense and collision signals). The TX Ethernet MAC is divided into several modules that provide the following functionality:

- Generation of the signals connected to the Ethernet PHY during the transmission process
- Generation of the status signals the host uses to track the transmission process
- Random time generation used in the back-off process after a collision has been detected
- CRC generation and checking
- Pad generation
- Data nibble generation

B.7.6 RX Ethernet MAC

The RX Ethernet MAC transmits the data streams to the host in response to the 10BASE-T or 100BASE-TX received MII nibbles. The module is divided into several sub-modules providing the following functionality:

- Preamble removal
- Data assembly (from input nibble to output byte)
- CRC checking for all incoming packets
- Generation of the signal that can be used for address recognition (in the hash table)
- Generation of the status signals the host uses to track the reception process

B.7.7 MAC Control Module

The MAC Control Module performs a real-time flow control function for the full-duplex operation. The control opcode PAUSE is used for stopping the station transmitting the packets. The receive buffer (FIFO) starts filling up when the upper layer cannot continue accepting the incoming packets. Before an overflow happens, the upper layer sends a PAUSE control frame to the transmitting station. This control frame inhibits the transmission of the data frames for a specified period of time.

When the MAC Control module receives a PAUSE control frame, it loads the pause timer with the received value into the pause timer value field. The Tx MAC is stopped (paused) from transmitting the data frames for the “pause timer value” slot times. The pause timer decrements by one each time a slot time passes by. When the pause time number equals zero, the MAC transmitter resumes the transmit operation.

The MAC Control Module has the following functionality:

- Control frame detection
- Control frame generation
- TX/RX MAC Interface
- PAUSE Timer
- Slot Timer

B.7.8 Control Frame Detection

The incoming data packets are passed from the receiver via the MAC Control Module to the upper layers while the control frames are usually dropped.

The RXFLOW bit in the CTRLMODER register defines whether the pause control frame causes the transmitter to break the transmission or not. If RXFLOW bit is set then the RXC interrupt is set in the INT_SOURCE register. The PASSALL bit in the CTRLMODER register defines whether the control frames are stored to the memory or not (regardless to the RXFLOW bit). If the PASSALL bit is set then the control frame is stored to the

memory and related buffer descriptor has the control frame bit (CF) set to 1. RXB interrupt in the INT_SOURCE register is set to 1.

Note: If both RXFLOW and PASSALL bits are set to 1, then only RXC interrupt is set in the INT_SOURCE register when a control frame is received.

A valid PAUSE control frame has the frame structure described in Table B.35:

Destination Address	Source Address	Length/Type	Opcode	Pause Timer Value	Reserved	CRC
6	6	2	2	2	42	4

Table B.35 Structure of the PAUSE control frame

The destination address must be a reserved multicast address (01-80-c2-00-00-01) or a destination address equal to the Ethernet IP Core MAC address. The Length/Type field must be equal to 8808 and the opcode to 0001 for a PAUSE control frame.

When the receive flow control and the MAC Control Module are enabled (RXFLOW asserted and PASSALL deasserted), a PAUSE Timer Value from the PAUSE control frame is passed to the PAUSE timer.

B.7.9 Control Frame Generation

When the host wants to send a PAUSE control frame, it asserts the Transmit Pause Request (TPAUSERQ). When a request is detected, the control module waits for the current transmission to end. It then starts transmitting the PAUSE control frame by asserting the Transmit Packet Start Frame (TxStartFrm) and providing the appropriate control data. Sending CtrlFrm is used to instruct the Transmit function (TX Ethernet MAC) to pad and append the FCS. The transmit Pause Frame End (TxEndFrm) is asserted at the end to inform the host that a Pause request was sent.

Asserting the TXFLOW bit in the CTRLMODER register enables the transmission of the PAUSE control frame.

When the control frame needs to be sent, the request is issued by writing the appropriate value to the TXCTRL register. The Transmit Pause Timer Value TPAUSETV[15:0] contains the value to be sent as a Pause Timer Value in the pause control frame (Table B.35). The TPAUSERQ signal (request) is latched internally in the MAC Control Generator and reset after the PAUSE control frame has been transmitted. This prevents issuing a new PAUSE request until the current request is sent.

B.7.10 TX/RX MAC Interface

The MAC Control Module is connected between the host and the Tx and Rx modules. When enabled, the its logic takes over the control of the following signals: TxData[7:0], TxStartFrm, TxEndFrm, TxUsedData, TxDone, and TxAbort. These signals are connected directly between the host and the MAC transmit and receive functions when data frames (not control frames) are transmitted or received.

On the other hand, when a host wants to send a PAUSE control frame, it asserts a TPauseRQ request signal. It is then up to the MAC Control Module to initiate the transmission. In this case, the above signals are not connected to the host any more. The MAC Control Module drives the appropriate control data signals and instructs the Tx module to transmit.

When a PAUSE control frame is received, the frame can be dropped or passed to the host, depending on the state of the PASSALL and RXFLOW bits (See CTRLMODER (Control Module Mode Register) in section B.4.4 for more details). Again TxData[7:0], TxStartFrm, TxEndFrm, TxUsedData, TxDone, and TxAbort are not connected directly.

B.7.11 PAUSE Timer

The 16bit PAUSE timer is loaded with a pause timer value when a PAUSE control frame is received. The timer inhibits the data frame transmissions for the timer value time slots. This is done by:

- Preventing the Tx MAC module from seeing the signal TxStartFrm from the host
- Preventing the host from seeing the signal TxUsedData from the Tx module

The timer decrements by one each time a time slot passes by. A Slot Timer is used for counting the slot time.

B.7.12 Slot Timer

The Slot Timer is activated when a PAUSE Timer is preloaded. It counts slot times and generates pulses to the PAUSE Timer for every slot time passed. Slot time is time, needed for transmission of the 64 bytes.

B.8 MII Management Module

The MII Management Module is a simple two-wire interface between the host and an external PHY device. It is used for configuration and status read of the physical device.

The physical interface consists of a management data line MDIO (Management Data Input/Output) and a clock line MDC (Management Data Clock). During the read/write operation, the most significant bit is shifted in/out first from/to the MDIO data signal. On each rising edge of the MDC, a Shift register is shifted to the left and a new value appears on the MDIO.

Internally the interface consists of four signals:

- MDC
- MDI
- MDO
- MDOEN (Management Data Output Enable)

The unidirectional lines MDI, MDO, and MDOEN are combined to make a bi-directional signal MDIO that is connected to the PHY.

The configuration and status data is written/read to/from the PHY via the MDIO signal.

The MDC is a low frequency clock derived from dividing the host clock.

Three commands are supported for controlling the PHY:

- Write Control Data (writes the control data to the PHY Configuration registers)
- Read Status (reads the PHY Control and Status register)
- Scan Status (continuously reads the PHY Status register of one or more PHYs [link fail status]).

The MII Management Module consists of four sub modules:

- Operation Controller
- Shift Registers
- Output Control Module
- Clock Generator

B.8.1 Operation Controller

The Operation Controller's task is to perform all supported commands: Write Control Data, Read Status, and Scan Status.

Write Control Data

A host initiates a write operation by asserting the WCTRLDATA signal. This signal also indicates that the host data CTLD[15:0], the PHY address FIAD[4:0], and the PHY register address RGAD[4:0] are valid. As soon as the host asserts the WCTRLDATA signal, the MIIM module asserts the BUSY signal to inform the host that the write operation is in process. MDOEN is asserted to enable the output line MDO (MDIO) to the PHY. The MIIM module then clocks out the MIIM frame to the PHY on each rising edge of the MDC. The MIIM frame write format conforms to the *IEEE 803.2u Specification*:

- 32-bit long preamble (all ones) if the MIINOPRE bit is not asserted
- 2-bit long Start of frame pattern ST (zero followed by one)
- 2-bit Operation definition (zero-one for write or one-zero for read)
- 5-bit PHY address (FIAD[4:0])
- 5-bit PHY register address RGAD[4:0]
- 2-bit turnaround field TA (one-zero)
- 16-bit data

At the end of the write operation, the BUSY signal is deasserted.

Read Status

A host initiates a write operation by asserting the RSTAT signal. This signal also indicates that the PHY address FIAD[4:0] and the PHY register address RGAD[4:0] are valid. As soon as the host asserts the RSTAT signal, the MIIM module asserts the BUSY signal to inform the host that the read operation is in process. MDOEN is asserted to enable the output line MDO (MDIO) to the PHY. The MIIM module then clocks out the MIIM frame to the PHY on each rising edge of the MDC and afterwards clocks in the requested data (status). The MIIM read frame format conforms to the *IEEE 803.2u Specification*:

- 32-bit long preamble (all ones) if the MIINOPRE bit is not asserted
- 2-bit long Start of frame pattern ST (zero followed by one)
- 2-bit Operation definition (zero-one for write or one-zero for read)
- 5-bit PHY address (FIAD[4:0])
- 5-bit PHY register address RGAD[4:0]
- 2-bit turnaround field TA (one-bit period in which the PHY stays in the high-Z state followed by a one-bit period during which the PHY drives a zero on the MDO)

- MIIM deasserts the MDOEN signal that enables the MDI (MDIO works as an input)
- PHY sends the data (status) back to the MIIM Module on the data lines PRSD[15:0]

At the end of the read operation, the MIIM deasserts the BUSY signal to indicate to the host that valid data is on the PRSD[15:0] lines.

Scan Status

A host initiates the Scan Status Operation by asserting the SCANSTAT signal. The MIIM performs a continuous read operation of the PHY Status register. The PHY is selected by the FIAD[4:0] signals. The link status LinkFail signal is asserted/deasserted by the MIIM module and reflects the link status bit of the PHY Status register. The signal NVALID is used for qualifying the validity of the LinkFail signals and the status data PRSD[15:0]. These signals are invalid until the first scan status operation ends.

During the scan status operation, the BUSY signal is asserted until the last read is performed (the scan status operation is stopped).

B.8.2 Shift Registers Operation

There are two shift registers in the MII Management Module. The Data Shift register is used for:

- Shifting out the data to the PHY during the Write Data Control operation
- Shifting in the data during the Read Status operation
- Shifting out the FIAD[4:0] and RGAD[4:0] addresses during all operations

The Status Shift register contains the data latched during the last Read Status Operation. Two additional status signals (LinkFail Status and Status Invalid NVALID) are latched separately from the Status Shift register.

When a Scan Operation is requested, the state of the PRSD[15:0] and a MIILF is constantly updated from the selected PHY register. NVALID is used to qualify the validity of the PRSD[15:0] and MIILS signals. These signals are invalid until the first Scan Status Operation ends.

B.8.3 Output Control Module Operation

The Output Control Module combines the MDI, MDO, and MDOEN signals into a bi-directional MDIO signal that is connected to the external MII PHY. During the Write Control Data Operation, the MDIO operates as an output from the MIIM module. The signal is used for transferring data from the MIIM Module to the PHY. During the Read

Status Operation, the MDIO first operates as an output (addressing the PHY and the PHY Internal register) and then as an input to the MIIM Module (reading the status data). In both cases the most significant bit of the data is shifted first. When no operation is performed, the MDIO is tri-stated.

B.8.4 Clock Generator Operation

The Management Data Clock MDC is a divided host clock. The division factor is set in the MIIMODER register by setting the CLKDIV[7:0] field (MDC depends on the PHY and can be 2.5 MHz or 12.5 MHz.

B.9 Architecture

The Ethernet IP Core consists of 5 modules:

- Host Interface and the BD structure
- TX Ethernet MAC (transmit function)
- RX Ethernet MAC (receive function)
- MAC Control Module
- MII Management Module

B.10 Host Interface

The host interface is connected to the RISC and the memory through the two Wishbone interfaces. The RISC writes the data for the configuration registers directly while the data frames are written to the memory. For writing data to configuration registers, Wishbone slave interface is used. Data in the memory is accesses through the Wishbone master interface.

B.11 TX Ethernet MAC

The TX Ethernet MAC generates 10BASE-T/100BASE-TX transmit MII nibble data streams in response to the byte streams supplied by the transmit logic (host). It performs the required deferral and back-off algorithms, takes care of the IPG, computes the checksum (FCS) and monitors the physical media (by monitoring Carrier Sense and collision signals).

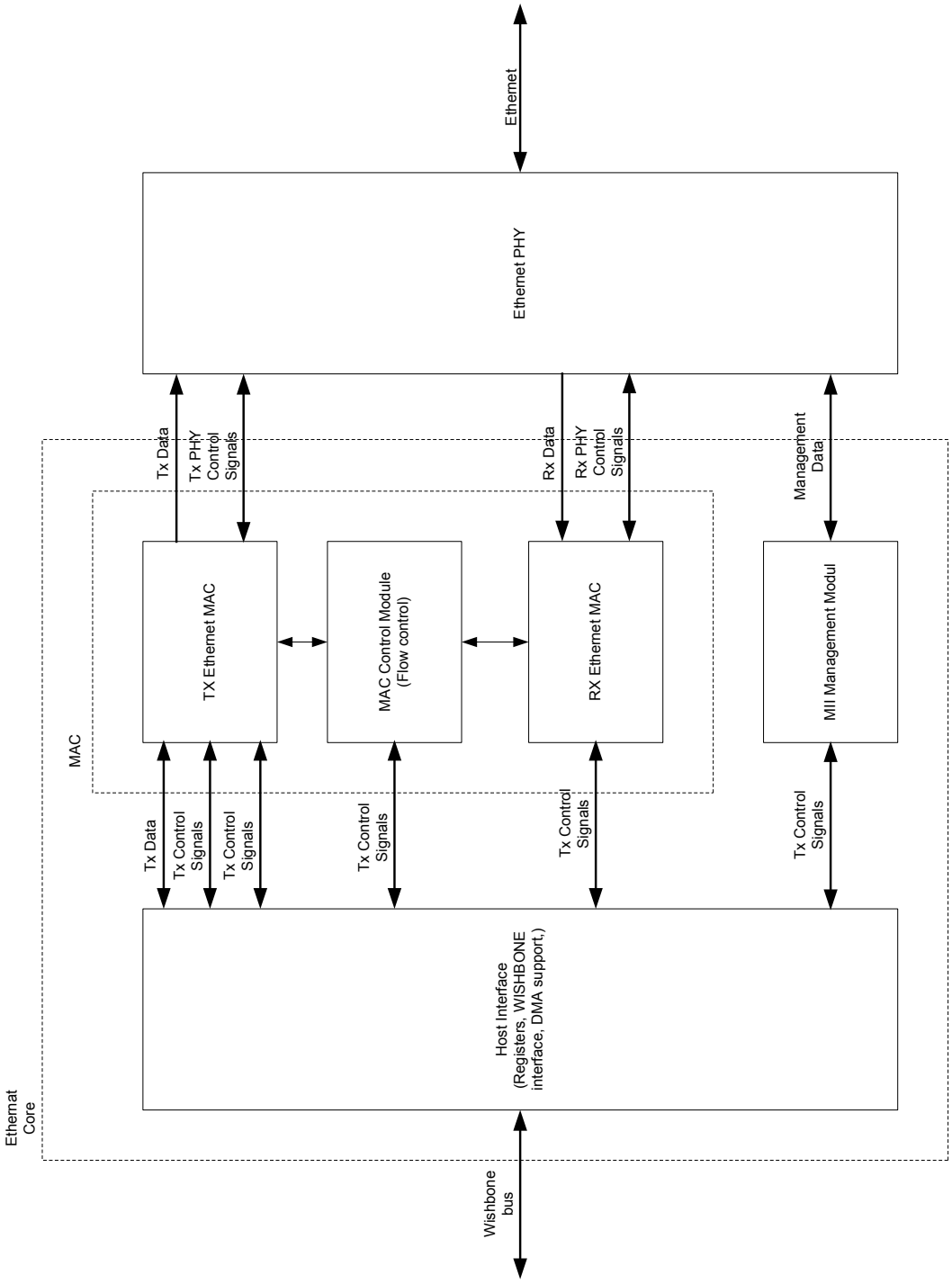


Figure B.2 Architecture Overview

B.12 RX Ethernet MAC

The RX Ethernet MAC interprets 10BASE-T/100BASE-TX MII receive data nibble streams and supplies correctly formed packet-byte streams to the host. It searches for the SFD (start frame delimiter) at the beginning of the packet, verifies the FCS, and detects any dribble nibbles or receive code violations.

B.13 MAC Control Module

The function of this module is to implement the full-duplex flow control.

The MAC Control Module consists of three sub-modules that provide the following functionality:

- Control frame detection
- Control frame generation
- TX/RX Ethernet MAC Interface
- PAUSE Timer
- Slot Timer

B.13.1 Control Frame Detector

The control frame detector checks the incoming frames for the control frames. Control frames can be discarded or passed to the host. When a PAUSE control frame is detected, it can stop the Tx module from transmitting for a certain period of time.

B.13.2 Control Frame Generator

If the need arises to stop the transmitting station from transmitting (flow control in full-duplex mode), a PAUSE control frame can be sent.

B.13.3 TX/RX Ethernet MAC Interface

The MAC Control Module is connected between the host interface, the Tx, and the Rx MAC modules. Signals from the host are passed to the Tx MAC in certain occasions and vice versa.

B.13.4 PAUSE Timer

When a PAUSE control frame is received, the pause timer value is written to the PAUSE timer. This prevents the Tx module from transmitting for a Âżpause timer valueÂń period of slot time.

B.13.5 Slot Timer

The slot timer measures time slots and generate a pulse to the PAUSE timer for every slot time passed by.

B.14 MII Management Module

The function of the MII Management Module is to control the PHY and to gather information from it (status).

It consists of four sub modules:

- Operation Control Module
- Output Control Module
- Shift Register
- Clock Generator

B.14.1 Operation Control Module

The function of the Operation Control Module is to perform the following commands:

- Write control data
- Read status
- Scan status

B.14.2 Output Control Module

The Output Control Module controls the signal appearance on the MDO, MCK, and MDOEN pins.

B.14.3 Shift Register

The shift registers hold the status read from an external PHY.

B.14.4 Clock Generator

The clock generator generates an appropriate output clock MCK according to the input host clock and the clock divider bits (CLKDIV[7:0] in the MIIMODER register).

LISTE DES RÉFÉRENCES

[1] (Juillet 2014). http://sipi.usc.edu/database/database.php?volume=misc.

[2] (Décembre 2014). http://www.fmwconcepts.com/imagemagick/gaussian/index.php.

[3] (Juillet 2014). http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/.

[4] (Juillet 2014). http://www.eng.utah.edu/~cs3710/xilinx-docs/wp231.pdf.

[5] (Juillet 2014). http://www.altera.com/literature/hb/qts/qts_qii51007.pdf.

[6] (Juillet 2014). http://webstaff.itn.liu.se/~qinye29/dce/vhdl_xilinx_coding.pdf.

[7] (Juillet 2014). http://dawsonjon.github.io/Chips-2.0/ (page consultée le 20 juillet).

[8] Abdelfattah, E. et Mohiuddin, A. (2010). Performance Analysis Of Multimedia Compression Algorithms. *International journal of computer science and information Technology (IJCSIT)*, volume 2, numéro 5, p. 1–10.

[9] Adams, M. (2014). The JasPer Project Home Page. http://www.ece.uvic.ca/~mdadams/jasper/.

[10] Adams, M. D. et Ward, R. (2001). Wavelet Transforms in the JPEG-2000 Standard. Dans *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. volume 1. p. 160–163.

[11] Ajmone, M., Corazza, M. G., Listanti, M. et (Eds.), A. R. (2003). *Improving End-to-End Performance in Reconfigurable Networks through Dynamic Setting of TCP Parameters*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[12] Ajmone, M., Corazza, M. G., Listanti, M. et (Eds.), A. R. (2003). *Live Admission Control for Video Streaming*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[13] Ajmone, M., Corazza, M. G., Listanti, M. et (Eds.), A. R. (2003). *TCP Smart Framing : A Segmentation Algorithm to Improve TCP Performance*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[14] Aktas, M. S., Fox, G. C. et Pierce, M. (2010). A Federated Approach to Information Management in Grids. *International Journal of Web Services Research*, volume 7, numéro 1, p. 65–98.

[15] Al-Canaan, A. et Khoumsi, A. (2008). Advanced IP-Telephony Service Creation Using JAIN-SIP API : Cross-Platform Approach. Dans *Mosharaka International Conference on Communications, Networking and Information Technology (MIC-CNIT 2008)*. p. 46–51.

[16] Al-Canaan, A. et Khoumsi, A. (2009). Analysis and Quantification of Multimedia Web Services Performance Utilising Binary Data Image Compression. Dans *The Third Mosharaka International Conference on Commuunications, Networking and Information Technology (MIC-CNIT 2009)*. p. 75–80.

[17] Al-Canaan, A. et Khoumsi, A. (2010). Cross-Platform Approach to Advanced IP-Telephony Services Using JAIN-SIP. *Journal of Networks*, volume 5, numéro 7, p. 8080–814.

[18] Al-Canaan, A. et Khoumsi, A. (2011). Multimedia Web Services Performance : Analysis and Quantification of Binary Data Compression. *Journal of Multimedia (JMM)*, volume 6, numéro 5, p. 447–457.

[19] Al-Canaan, A. et Khoumsi, A. (2011). Performance Enhancement of Image-Retrieval web Services Through Image Dimensional Optimisation. Dans *Mosharaka International Conference on Wireless Communications and Mobile Computing (MIC-WCMC2011)*. p. 45–50.

[20] Al-Canaan, A. et Khoumsi, A. (2011). The Impact of Binary Data Compression on QoS and Performance of SOAP and RESTful Multimedia Web Services. Dans *Mosharaka International Conference on Wireless Communications and Mobile Computing (MIC-WCMC2011)*. p. 77–83.

[21] Aloqaily, M., Belqasmi, F., Glitho, R. et Hammad, A. (2013). Multiparty/multimedia conferencing in mobile ad hoc networks for improving communications between firefighters. Dans *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on*. p. 1–7.

[22] Antonini, M., Barlaud, M., Mathieu, P. et Daubechies, I. (1992). Image Coding Using Wavelet Transform. *IEEE Transactions on image processing*, volume 1, numéro 2, p. 205–220.

[23] Artail, H. et Kahale, E. (2006). Maws : A platform-independent framework for mobile agents using web services. *J. Parallel Distrib. Comput.*, volume 66, numéro 3, p. 428–443.

[24] Asprino, P., Fresa, A., Gaito, N. et Longo, M. (2003). A Layered Architecture to Manage Complex Multimedia Services. *Software Engineering and Knowledge Engineering*, p. 414–421.

[25] Baldo, N., Horn, U., Kampmann, M. et Hartung, F. (2004). Rtcp feedback based transmission rate control for 3g wireless multimedia streaming. Dans *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications 2004 (PIMRC2004*.

[26] Balk, A., Maggiorini, D., Gerla, M. et Sanadidi, M. (2003). *Adaptive MPEG-4 Video Streaming with Bandwidth Estimation*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[27] Belqasmi, F., Singh, J., Bani Melhem, S. et Glitho, R. H. (2012). Soap-based vs. restful web services : A case study for multimedia conferencing. *IEEE Internet Computing*, volume 16, numéro 4, p. 54–63.

[28] Bond, M. (2001). Attacks on Cryptoprocessor Transaction Sets. Dans *CHES 2001, Springer LNCS*. Springer-Verlag, p. 220–234.

[29] Buraga, S. C. (2003). A Distributed Platform based on Web Services for Multimedia Resource Discovery. Dans *Proceedings of the 2nd International Symposium on Paral lel and Distributed Computing, IEEE Computer*. Society Press.

[30] Cai, M., Ghandehaizadeh, S., Schmidt, R. et Song, S. (2002). *A Comparison of Alternative Encoding Mechanisms for Web Services*, Lecture Notes in Computer Science, *volume 2453/2002*. Springer, Berlin.

[31] Calarco, G., Maccaferri, R., Pau, G. et Raffaelli, C. (2003). *Design and Implementation of a Test Bed for QoS Trials*, Berlin Heidleberg. volume QoS-IP. Springer.

[32] Chandra, S., Ellis, C. et Vahdat, A. (2000). Differentiated multimedia web services using quality aware transcoding. Dans *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. volume 2. p. 961–969 vol.2.

[33] Chandra, S., Ellis, C. S. et Vahdat, A. (1999). Multimedia web services for mobile clients using quality aware transcoding. Dans *Proceedings of the 2Nd ACM International Workshop on Wireless Mobile Multimedia*. WOWMOM '99. ACM, New York, NY, USA, p. 99–108.

[34] Chandra, S., Ellis, C. S. et Vahdat, A. (2000). Differentiated multimedia web services using quality aware transcoding.

[35] Channabasavaiah, K., Holley, K. et Jr., E. T. (2006). Migrating to a service-oriented architecture part 1.

[36] Che, S., Li, J., Sheaffer, J., Skadron, K. et Lach, J. (2008). Accelerating Compute-Intensive Applications with GPUs and FPGAs. Dans *Application Specific Processors, 2008. SASP 2008. Symposium on*. p. 101–107.

[37] Chen, L. et Li, C. (2007). Design and Implementation of the Network Server Based on SIP Communication Protocol. *World Academiy of science, Engineering and Technology*, p. 485–488.

[38] Clayton, R. et Bond, M. (2003). Experience Using a Low-Cost FPGA Design to Crack DES Keys. Dans *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, volume 2523 of series, pages 579 – 592*. Springer-Verlag, p. 579–592.

[39] Cook, W. R. et Barfield, J. (2006). Web Services versus Distributed Objects : A Case Study of Performance and Interface Design. Dans *Proceedings of the IEEE International Conference on Web Services*. IEEE Computer Society, Washington, DC, USA, p. 419–426.

[40] Costa, R., Alves, G., Zenha-Rela, M., Poley, R. et Wishart, C. (2009). FPGA-based Weblab Infrastructures Guidelines and a Prototype Implementation Example. Dans *E-Learning in Industrial Electronics, 2009. ICELIE '09. 3rd IEEE International Conference on*. p. 57–63.

[41] Cruz-lara, S., Ducret, J. et Faivre-vuillin, J. (2003). Using webservices for accessing and sharing multimedia resources on a distributed software architecture. Dans *in IEEE Multimedia Software Engineering 2003 - MSE 2003, Taichung, Taiwan, IEEE*.

[42] Data, R. (2006). SOAP Tutorial.

[43] Deng, Y., Sadjadi, M., Clarke, P. J., Zhang, C., Rangaswami, R. et Prabakar, N. (2006). A Communication Virtual Machine. Dans *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*. p. 521–531.

[44] Developer, S. (2009). Metro Web Services Technologies. URL, http://java.sun.com/webservices/technologies/index.jsp.

[45] DjVu.org (2009). Downloads and Resources - DjVu.org. URL, http://djvu.org/resources/.

[46] Dutta, A., Fifield, J., Schelle, G., Grunwald, D. et Sicker, D. (2008). An Intelligent Physical Layer for Cognitive Radio Networks. Dans *Proceedings of the 4th Annual International Conference on Wireless Internet*. WICON '08. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, p. 12 :1–12 :9.

[47] Falcarin, P. (2004). A CPL to Java Compiler for Dynamic Service Personalization in JAIN-SIP Server. *In IEC Annual Review of Communications*, volume 57, p. 577–586.

[48] FMJ (2009). Freedom for Media in Java. URL, http://fmj-sf.net/.

[49] Fowers, J., Brown, G., Cooke, P. et Stitt, G. (2012). A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications. Dans *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. FPGA '12. ACM, New York, NY, USA, p. 47–56.

[50] Fund, F., Wang, C., Liu, Y., Korakis, T., Zink, M. et Panwar, S. (2013). Performance of DASH and WebRTC Video Services for Mobile Users. Dans *Packet Video Workshop (PV), 2013 20th International*. p. 1–8.

[51] Ghandeharizadeh, S., Papadopoulos, C., Pol, P. et Zhou, R. (2003). Nam : a network adaptable middleware to enhance response time of web services. Dans

Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on. p. 136–145.

[52] Giordano, S., Listanti, M., Mustacchio, F., Niccolini, S., Salsano, S. et Veltri, L. (2003). *SIP Originated Dynamic Resource Configuration in DiffServ Networks : SIP / COPS / Traffic Control Mechanisms*, Berlin Heidleberg. volume QoS-IP. Springer.

[53] Goth, G. (2004). Critics Say Web Services Need a REST. *IEEE Distributed Systems Online*, volume 5, numéro 12.

[54] Gottschalk, K., Graham, S., Kreger, H. et Snell, J. (2002). Introduction to Web Services architecture. *IBM SYSTEMS JOURNAL*, volume 41, numéro 2, p. 170–177.

[55] Grafl, M., Timmerer, C., Hellwagner, H., Cherif, W., Negru, D. et Battista, S. (2013). Scalable Video Coding Guidelines and Performance Evaluations for Adaptive Media Delivery of High Definition Content. Dans *Proceedings of the 18th IEEE International Symposium on Computers and Communication (ISCC 2013)*.

[56] Greco, R., Delgrossi, L. et Brunner, M. (2003). *Towards RSVP Version 2*, Berlin Heidleberg. volume QoS-IP. Springer.

[57] Haffner, P., Bottou, L., yann LeCun et Vincent, L. (2001). A General Segmentation Scheme for DjVu Document Compression. Dans *ISMM'02*.

[58] Hamad, H., Saad, M. et Abed, R. (2010). Performance evaluation of restful web services for mobile devices. *Int. Arab J. e-Technol.*, volume 1, numéro 3, p. 72–78.

[59] Herreria-Alonso, S., Suarez-Gonzalez, A., Fernandez-Veiga, M., Rodriguez-Rubio, R. F. et Lopez-Garcia, C. (2003). *Simulation Study of Aggregate Flow Control to Improve QoS in a Differentiated Services Network*, Berlin Heidleberg. volume QoS-IP. Springer.

[60] Hopkins, M. et Brown, L. (2008). An Introduction to QuickTime for Java Building a Zoo Kiosk. URL, http://developer.apple.com/quicktime/qtjava/qtjtutorial/index.html.

[61] Hormati, M., Belqasmi, F., Glitho, R. et Khendek, F. (2013). A DNS protocol - based Service Discovery architecture for disaster response systems. Dans *Computers and Communications (ISCC), 2013 IEEE Symposium on*. p. 000366–000371.

[62] Huang, J. et Lin, C. (2013). Improving Energy Efficiency in Web Services : An Agent-Based Approach for Service Selection and Dynamic Speed Scaling. *Int. J. Web Serv. Res.*, volume 10, numéro 1, p. 29–52.

[63] inc., D. (Juillet 2014). http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,897&Prod=NEXYS3.

[64] Iren, S., Amer, P. D. et Conrad, P. T. (1999). The Transport Layer :Tutorial and Survey. *ACM Computing Surveys*, volume 31, numéro 4, p. 360–405.

[65] Jaakko, K. (2007). Efficient Implementation of XML Security for Mobile Devices. Dans *IEEE International Conference on Web Services (ICWS 2007)*.

[66] KHALSA, N., SARATE, G. G. et INGOLE, D. T. (2010). Factors Influencing The Image Compression of Artificial and Natural Image Using Wavelet Abstract : Transform. *International Journal of Engineering Science and Technology*, volume 2, numéro 11, p. 6225–6233.

[67] Király, C., Pándi, Z. et Do, T. V. (2003). *Analysis of SIP, RSVP, and COPS Interoperability*, Berlin Heidleberg. volume QoS-IP. Springer.

[68] Kostoulas, M., Matsa, M., Mendelsohn, N., Perkins, E., Heifets, A. et Mercaldi, M. (2006). Xml Screamer : An Integrated Approach to High Performance XML Parsing, Validation and Deserialization. Dans *15th International WWW Conference*. ACM Press.

[69] Kuon, I. et Rose, J. (2007). Measuring the Gap Between FPGAs and ASICs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, volume 26, numéro 2, p. 203–215.

[70] Kuon, I. et Rose, J. (2007). Measuring the Gap Between FPGAs and ASICs. *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, volume 26, numéro 2, p. 203–215.

[71] Lee, S., Ryu, K. D., Lee, K.-W. et Choi, J.-D. (2007). Improving the Performance of Web Services Using Deployment-Time Binding Selection. Dans *IEEE International Conference on Web Services (ICWS 2007)*, IEEE.

[72] Leighton, T. (2009). Improving performance on the internet. *Commun. ACM*, volume 52, numéro 2, p. 44–51.

[73] Licciardi, C. A. et Falcarin, P. (2003). Technologies and Guidelines for Service Creation in NGN. *exp*, volume 3, numéro 4, p. 46–53.

[74] LLC, I. S. (2009). ImageMagick. URL, http://www.imagemagick.org/script/index.php.

[75] Lu, F. et Chia, L.-T. (2006). Multimedia Web Services for an Object Tracking and Highlighting Application. Dans Cham, T.-J., Cai, J., Dorai, C., Rajan, D., Chua, T.-S. et Chia, L.-T., *Advances in Multimedia Modeling*, Lecture Notes in Computer Science, *volume 4352*. Springer Berlin Heidelberg, p. 238–247.

[76] Malhotra, R., Gupta, V. et Bansal, D. R. K. (2011). Simulation and Performance Analysis of Wired and Wireless Computer Networks. *International Journal of Computer Applications*, volume 14, numéro 7, p. 11–17.

[77] Mani, A. et Nagarajan, A. (2002). Understanding Quality of Service for Web Services. URL, http://www.ibm.com/developerworks/library/ws-quality.html.

[78] Martin, T. A. et Davis, G. (1997). *The Project Cool Guide to HTML*. Katherine Schowalter.

[79] Matei, R. (2002). JAIN-SIP Approach in Ad-Hoc Networks Mobility Management. *Ad Hoc Mobile Wireless Networks*.

[80] McCanne, S., Jacobson, V. et Vetterli, M. (1996). Receiver-driven layered multicast. p. 117–130.

[81] McDonnell, R. et McNamara, A. (2003). Application of the Golden Ratio to 3D Facial Models. Dans *Proceedings Eurographics Ireland*. p. 35–42.

[82] Microsystems, S. (2008). Java Media Framework API (JMF). URL, http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/download.html.

[83] Mitzenmacher, M. (2001). On the Hardness of Finding Optimal Multiple Preset Dictionaries. Dans *Proceedings DCC 2001. Data Compression Conference*. p. 411–418.

[84] Modayil, J., Cheng, H. et Li, X. (1997). Experiments in Simple One-Dimensional Lossy Image Compression Schemes. *Multimedia Computing and Systems, International Conference on*, volume 0, p. 614.

[85] Moffat, A., Bell, T. et Witten, I. (1997). Lossless Compression for Text and Images. *International Journal of High-Speed Electronics and Systems*, volume 8, numéro 1, p. 179–231.

[86] Murray, J. D. et Van Ryper, W. (2011). *Encyclopedia of Graphics File Formats*, 2e édition. O'REILLY and Associates, inc.

[87] Nayak, D., Phatak, D. B. et Saxena, A. (2008). Evaluation of Security Architecture for Wireless Local Area Networks by Indexed Based Policy Method : A Novel Approach. *International Journal of Network Security*, volume 7, numéro 1, p. 1–14.

[88] netbeans.org (2009). End-to-End Web Service Tutorial : Flower Application. URL, http://www.netbeans.org/kb/60/websvc/ejb.html.

[89] Ng, A., Greenfield, P. et Chen, S. (2005). A study of the impact of compression and binary encoding on soap performance. *Proceedings of the Sixth Australasian Workshop on Software and System Architectures (AWSA2005)*, p. 46–56.

[90] Nwak, S., Domanska, J. et Domanski, A. (2009). Simulation Models of Fair Scheduling for the TCP and UDP Streams. *Theoretical and Applied Informatics*, volume 21, numéro 3-4, p. 193–204.

[91] Onose, N., Khalaf, R., Rose, K. et Siméon, J. (2008). A Restful Workflow Implementation on Top of Distributed XQuery. Dans *5th Int'l Workshop on XQuery Implementation, Express and Perspectives (XIME-P 2008)*.

[92] Phirke, V., Claypool, M. et Kinicki, R. (2003). *Traffic Sensitive Active Queue Management for Improved Multimedia Streaming*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[93] Pujolle, G. et E.Horlait (1990). *Architectures des Réseaux Informatiques Tome 1 Les outils de communications*. EYEROLLES, Paris.

[94] Recker, S., Lüdiger, H. et Geisselhardt, W. (2003). *Dynamic Adaptation of Virtual Network Capacity for Deterministic Service Guarantees*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[95] Richardson, L. et Ruby, S. (2007). *RESTful Web Services*. O'REILLY.

[96] Rosario, S., Benveniste, A., Haar, S. et Jard, C. (2008). Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations. Dans *IEEE Transactions on Services Computing*. IRISA/INRIA, Rennes, p. 187–200.

[97] Ruta, A., Brzoza-Woch, R. et Zielinski, K. (2012). On Fast Development of FPGA-based SOA Services—Machine Vision Case Study. *Design Automation for Embedded Systems*.

[98] S., E., Atkinson, R., Castro, D. et G., O. (2009). The Need for Speed : The Importance of Next-Generation Broadband Networks. *The information technology & innovation foundation*, p. 1–38.

[99] Scholz, A., Buckl, C., Kemper, A., Knoll, A., Heuer, J. et Winter, M. (2008). WS-AMUSE - Web Service Architecture for Multimedia Services. Dans *Proceedings of the 30th International Conference on Software Engineering*. ICSE '08. ACM, New York, NY, USA, p. 703–712.

[100] Schreiber, D., Göb, A., Aitenbichler, E. et Mühlhäuser, M. (2011). Reducing User Perceived Latency with a Proactive Prefetching Middleware for Mobile SOA Access. *Int. J. Web Serv. Res.*, volume 8, numéro 1, p. 68–85.

[101] Schulzrinne, H. (2008). Session Initiation Protocol (SIP). URL, http://www.cs.columbia.edu/sip/.

[102] search engine.com, P. (2009). Pdf Search Engine. URL, http://www.pdf-search-engine.com/jni-pdf.html.

[103] Shay, W. A. (1995). *Understanding Data Communications and Networks*. PWS.

[104] Singh, A., Mahadevan, P., Acharya, A. et Shae, Z. (2004). Design and Implementation of SIP Network and Client Services. *ICCCN*.

[105] SIP-Communicator.org (2008). SIP Communicator. URL, http://sip-communicator.org/.

[106] Spero, S. E. (2014). Understanding Quality of Service for Web Services. URL, http://www.ibiblio.org/mdma-release/http-prob.html.

[107] Stuedi, P., Bihr, M., Remund, A. et Alonso, G. (2007). SIPHoc : Efficient SIP Middleware for Ad Hoc Networks. Dans *ACM/IFIP/USENIX 8th International Middleware Conference (Middleware 2007)*. p. 26–30.

[108] Sun, Y., Qiao, X., Cheng, B. et Chen, J. (2013). A Low-Delay, Lightweight Publish/Subscribe Architecture for Delay-Sensitive IOT Services. Dans *Web Services (ICWS), 2013 IEEE 20th International Conference on*. p. 179–186.

[109] Tian, M., Gramm, A., Ritter, H., Schiller, J. H. et Voigt, T. (2004). QoS-aware Cross-layer Communication for Mobile Web Services with the WS-QoS Framework. Dans *GI Jahrestagung (2)*. p. 286.

[110] Tian, M., Voigt, T., Naumowicz, T., Ritter, H. et Schiller, J. (2004). Performance considerations for mobile web services. *Comput. Commun.*, volume 27, numéro 11, p. 1097–1105.

[111] Tian, M., Voigt, T. et Ritter, H. (2003). Performance Impact of Web services on Internet Servers. Dans *Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems*.

[112] Vasudevan, V. (2001). A Web Services Primer.

[113] Vilas, J. F., Arias, J. J. P. et Vilas, A. F. (2006). Optimizing Web Services Performance Using Cache. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, volume 10, numéro 5, p. 713–720.

[114] W3C (2001). Web Services Description Language (WSDL) 1.1.

[115] W3C (2004). Web Services Description Language (WSDL) Version 2.0 Part 1 : Core Language.

[116] Wakamiya, N., Miyabayashi, M., Murata, M. et Miyahara, H. (2003). *Dynamic Quality Adaptation Mechanisms for TCP-friendly MPEG-4 Video Transfer*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[117] Watson, A. (1994). Image Compression Uing the Discrete Cosine Transform. *Mathematica Journal*, volume 4, numéro 1, p. 81–88.

[118] Werner, C., Buschmann, C. et Brandt, Y. F. S. (2008). XML Compression for Web Services on Resource-Constrained Devices. *International Journal of Web Services Research*, volume 5, numéro 3, p. 44–63.

[119] Werner, C., Buschmann, C., Jäcker, T. et Fischer, S. (2006). Bandwidth and Latency Considerations for Efficient SOAP Messaging. *International Journal of Web Services Research*, volume 3, numéro 1, p. 49–67.

[120] Wikipedia (2007). Representational State Transfer.

[121] Wu, P., Xu, J., Chen, F., Liu2, J., Zhang3, J. et Li, B. (2003). *On the Use of Sender Adaptation to Improve Stability and Fairness for Layered Video Multicast*, Verlag. volume QoS-IP. Springer, Berlin Heidleberg.

[122] Ye, J., Janardan, R. et Li, Q. (2004). GPCA : An Efficient Dimension Reduction Scheme for Image compression and Retrieval. Dans *KDD'04*.

[123] Yu, T. et kwei Jay Lin (2004). The Design of QoS Broker Algorithms for QoS-Capable Web Services. *International Journal of Web Services Research*, volume 1, numéro 33, p. 33–50.

[124] Zhang, G. et M., H. (2004). Implementing SIP and H.323 Signalling as Web Services. Dans *Euromicro Conference, 2004. Proceedings. 30th*. p. 265–271.

[125] Zhang, J. et Zhang, L.-J. (2005). Web Services Quality Testing. *International Journal of Web Services Research*, volume 2, numéro 2, p. 1–4.

[126] Zhang, J., Zhang, L.-J., Quek, F. et Chung, J.-Y. (2005). A Service-Oriented Multimedia Componentization Model. *International Journal of Web Services Research*, volume 2, numéro 1, p. 54–77.

[127] Zhou, C., Chia, L.-T. et Lee., B.-S. (2004). QoS-Aware and Federated Enhancement for UDDI. *International Journal of Web Services Research*, volume 1, numéro 2, p. 58–86.

[128] Zhou, C., Chia, L.-T. et Lee., B.-S. (2005). Web Services Discovery with DAML-QoS Ontology. *International Journal of Web Services Research*, volume 2, numéro 2, p. 43–67.

[129] Zhou, S. et Cheung, J. (2003). A Simple Platform independent Video/Voice over IP Application. *Australian Telecommunication Networks and Applications Conference 2003 (ATNAC 2003)*.

[130] zur Muehlen, M., Nickerson, J. V. et Swenson, K. D. (2005). Developing Web Services Choreography Standards - The Case of REST vs. SOAP. *Decision Support Systems*, volume 40, numéro 1, p. 9–29.